Investigation of MAML for Few-Shot Deep Reinforcement Learning

Kevin Hu
UC Berkeley
kevin.hu@berkeley.edu

Brian Wu
UC Berkeley
wubrian@berkeley.edu

Extended Abstract

In this project, we aim to explore how meta-learning can be applied to reinforcement learning tasks (Meta-RL). Among the many meta-learning approaches, we chose to use a gradient-based meta-RL algorithm known as Model-Agnostic Meta Learning (MAML). The primary idea behind MAML is to train our model on various training tasks, then update it based on how well it was able to adapt to the sampled training tasks in a few gradient steps. This allows us to essentially train a good initialization for our model parameters with which we hope that, when learning on new tasks similar to those seen during meta-training, the model will only have to make a few gradient steps, leading to much faster adaptation.

One crucial part of MAML is the policy gradient used in the meta-optimization step. We experimented with three different policy gradient algorithms for this part (VPG, TRPO, and PPO) and found that the policy gradient chosen had a large impact on our results. We also examined the effect of other parameters, such as the number of gradient (adapt) steps.

In addition to gaining insight into MAML, we wanted to use it on the OpenAI Retro Contest to compare with other algorithms. We ultimately found that meta-learning does not seem to do particularly well for this scenario. The algorithm requires much more memory than the baselines provided by OpenAI, and due to hardware limitations, we had to made adjustments to the large state space for MAML to run on the environment in a reasonable amount of time on our hardware. Evaluating the other algorithms, PPO2 in particular, with this downsampled state space resulted in MAML still underperforming for this task.

It would be worth exploring future experiments in the Gym Retro environment using MAML with alternative policy gradient algorithms or with more computation- and memory-efficient implementations.

1 Introduction

While many machine learning algorithms can achieve impressive results with large amounts of data, we may often encounter situations where we only have access to small datasets for particular tasks. We would also hope that our learned models would be able to leverage prior experience on different tasks in generalizing to new tasks, as humans do. Moreover, the ability to generalize can be seen as a measure of intelligence, one that we aim to train in artificial agents.

This makes few-shot learning an interesting and important application to explore. In the context of deep reinforcement learning (RL), this translates to attempting to make the learning process more efficient for a new task after already learning on a series of previous tasks. This idea of learning to learn is known as meta-learning, and in the context of RL tasks, Meta-RL.

There have been many different interesting approaches to the problem that aim to gain a different type of insight from previous experience, such as meta-learning exploration strategies (Gupta et al. [2018]) and loss functions (Houthooft et al. [2018]). Of course, since our final goal is to learn a policy, there has also been work in meta-learning the policy parameters directly through optimization techniques. The main work in this area, which we build upon, is Model-Agnostic Meta Learning (MAML), a gradient-based Meta-RL algorithm that optimizes to learn a set of parameters which can easily adapt to new tasks (Finn et al. [2017]).

Our primary focus in this project is to investigate the challenges and potential improvements to MAML in applying it to few-shot RL tasks. We tried to understand what makes meta-learning hard to use, and found that among the various parameters to tune and modifications to make, changing the policy gradients used in the meta-optimization had a large effect on the final result. We also wanted to attempt to tune MAML to tackle a more challenging application. To this end, we used the *Sonic the Hedgehog* games in the OpenAI Retro Contest (Nichol et al. [2018]) as a much harder benchmark for the generalization performance of our MAML models. However, we found that our results on this transfer learning contest were not as good as we hoped.

2 Gradient-based Meta-RL

2.1 Meta-RL

The meta-learning problem setup, as described in Finn et al. [2017], entails training a model f to map inputs x to outputs a where we train the model on a series of tasks, then train the model on new tasks, with the goal of adapting to these newer tasks faster. Of course, if the tasks seen during training time are not similar to the tasks at test time, it may be much harder to meta-learn a policy from which we can quickly adapt to new tasks, so we make the assumption of a distribution over tasks $p(\mathcal{T})$, where each task $\mathcal{T}$ is defined by $\mathcal{T} = \{\mathcal{L}(x_1, a_1, ..., x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H\}$ with loss function $\mathcal{L}$, initial distribution of observations $q(x_1)$, transition distribution $q(x_{t+1}|x_t, a_t)$, and episode lengths of H.

In the context of Meta-RL, each task $\mathcal{T}_i$ we sample is itself a Markov decision process (MDP) defined by a state space S, action space A, transition probabilities P_i, and reward function R_i. Note that each task shares the same state space and action space. In our meta-learning formulation, the loss functions $\mathcal{L}_{\mathcal{T}_i}$ are just the negation of the reward functions R_i.

For few-shot learning scenarios, we consider learning from only $\mathcal{K}$ samples of each new task $\mathcal{T}_i$, where each sample is drawn from the initial distribution $q_i(x_1)$ with trajectory drawn from transition distribution $q_i(x_{t+1}|x_t, a_t)$. This part of the process is known as meta-training, wherein we use our $\mathcal{K}$ samples of a particular task, along with that task's loss function $\mathcal{L}_i$, to train our model, then evaluate our model on new samples from the same task $\mathcal{T}_i$. This new data comes from the same distributions q_i, and we treat this test error on a particular training task as the training error in the overall meta-learning objective.

Once we complete our meta-learning process on the training tasks, we sample new tasks for meta-testing. Crucially, these new tasks come from the same distribution over tasks $p(\mathcal{T})$. This allows us to evaluate the ability of our trained model to quickly generalize to new tasks.

2.2 Model-Agnostic Meta-Learning

Model-Agnostic Meta-Learning (MAML) presents a direct approach to training our meta-learner (Finn et al. [2017]). Intuitively, our goal in Meta-RL is to train a policy that can adapt to new tasks faster by training on similar tasks. MAML is a gradient-based algorithm that sets up precisely this goal as an optimization problem, such that we can learn a set of model weights θ that parameterize our model f_θ. We hope that the learned model parameters serve as an improved initialization for adaptation on new tasks, such that they can achieve good results in a few gradient steps.

From this intuition we can formulate our meta-objective as:

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$

Concretely, after initializing our model parameters θ, we set up an outer meta-optimization loop where in each iteration we sample some batch of tasks $\mathcal{T}_i$ from $p(\mathcal{T})$. For each task $\mathcal{T}_i$ that we sample, we train (adapt) using an inner optimization loop with few-shot learning by sampling $\mathcal{K}$ trajectories with our model f_θ. With these trajectories, along with the task's loss function $\mathcal{L}_{\mathcal{T}_i}$, we are able to perform gradient descent on the model parameters:

$$\theta_i' \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_i(f_\theta)$$

This update represents one adaptation step; we can of course perform multiple adaptation steps, and in Section 3.1 we explore the performance results of doing so.

Once we finish computing a set of updated model parameters θ_i' corresponding to each task $\mathcal{T}_i$ in a batch, we perform meta-optimization on our model parameters θ with the following update rule:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$

In the two updates above, α and β are hyperparameters representing the step size and meta step size, respectively.

In practice, however, in the context of RL tasks, we cannot always easily solve for the gradient of the expected rewards. Instead, policy gradients are often employed to perform these updates directly to the policies. As in Finn et al. [2017], both our inner optimization and meta-optimization are performed with policy gradients. In our experiments, one of the primary factors we explored was the use of different policy gradients in computing the meta-optimization updates.

2.3 Policy Gradients

Policy gradients are algorithms that aim to directly maximize expected reward (or minimize loss) by using the gradient of the objective with respect to the model parameters θ. In MAML, policy gradients are used in the meta-optimization step, and we found that using different policy gradient algorithms changed results drastically. For our experiments we compared three on-policy algorithms: vanilla policy gradients, trust-region policy optimization (TRPO), and proximal policy optimization (PPO). We chose VPG to use as a simple baseline, TRPO because it was used in the original experiments done by Finn et al. [2017], and PPO because it is a popular and more recent algorithm.

Vanilla Policy Gradients In VPG, we run the policy and approximate the gradient of the expected reward $J(\theta)$ with respect to the model parameters:

$$\nabla_\theta J(\theta) \approx \sum_i (\sum_t \nabla_\theta \log \pi_\theta(a_t^i | s_t^i))(\sum_t r(s_t^i, a_t^i))$$

We then use this estimation in our standard update rule $\theta \leftarrow \alpha \nabla_\theta J(\theta)$

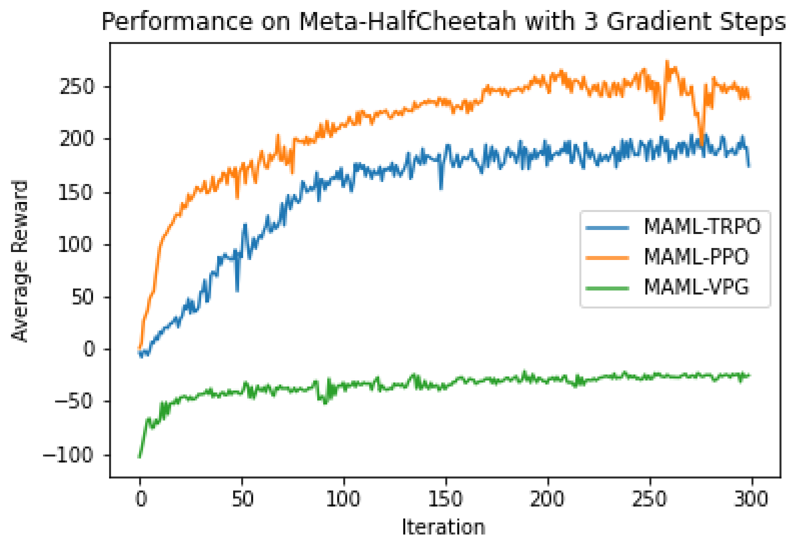
TRPO In TRPO (Schulman et al. [2017a]), the idea is to try to still improve model performance, but without taking gradient steps that change the policies too much. This is enforced with a constraint similar to KL-Divergence, which aims to limit the size of each update.

PPO PPO (Schulman et al. [2017b]) approaches the policy gradient with similar ideas of constraining the policy updates, but instead of being based on KL-Divergence, uses clipping on the objective.

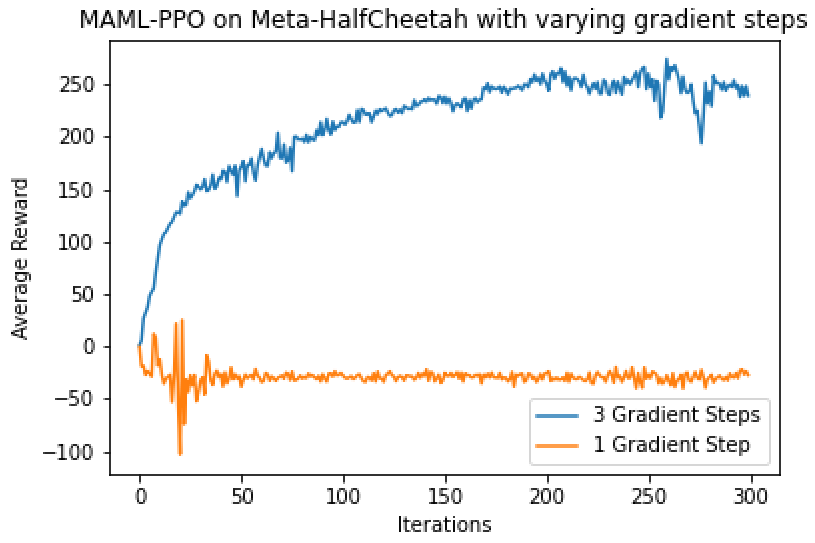
3 Experiments

3.1 Mujoco HalfCheetah

We looked at MAML's performance on a simple meta-learning task, implementing the MuJoCo Half-Cheetah Forward/Backward environment described in Finn et al. [2017], and comparing the performance of MAML-VPG, MAML-TRPO, and MAML-PPO. This environment selects a goal direction (-1 or 1) with equal probability and rewards the agent based on the velocity in the selected direction. The model trained by MAML in both cases is a MLP with two hidden layers of size 100, and uses 20 rollouts per gradient step, for 3 gradient steps.



We found that VPG performed much worse than the other two, while PPO outperformed TRPO. This was mostly expected, as VPG is much simpler than the other two, while we found TRPO to be harder to train.



We also experimented with tuning the number of adapt steps (gradient steps) during the training process. The general trend we found was that increasing the number of gradient steps improves performance, at the cost of training time.

3.2 Gym Retro Environments

OpenAI has previously released a set of baselines for various algorithms regarding few-shot learning in the Gym Retro environment (Nichol et al. [2018]), based on Sonic the Hedgehog levels from games on the SEGA Genesis. We sought to determine how well MAML would perform for these tasks.

3.2.1 Modifications

Due to hardware limitations, we were unable to run MAML with optimal parameters. Specifically, we made the following adjustments to allow us to use MAML with the Sonic environments:

Image Warping and Frame Stack The baselines provided by OpenAI warp the emulator's 224x320x3 images into a 84x84x4 stack of 4 grayscale images. We consistently ran into memory issues with this state size, so we chose to not use the frame stack and to downsample the images to 42x42. For the PPO2 baseline, this affected performance (run on validation level *GreenHillZone Act2* for 200k timesteps):

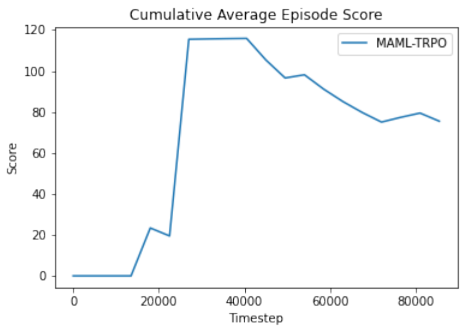
State Size	PPO2 Average Reward
84x84x4	2477.6 ± 435.3
42x42	1992.5 ± 750.1

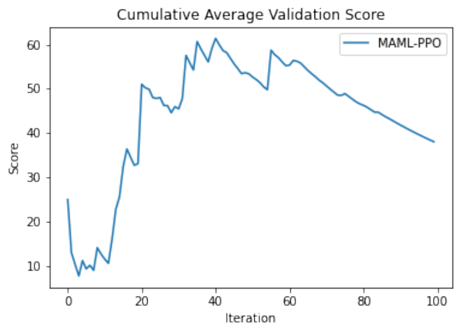
Policy For PPO2 on the *GreenHillZone Act2* level, we compared the performance of a CNN versus a MLP with two hidden layers of size 100. For our experiments involving MAML, we used the MLP policy.

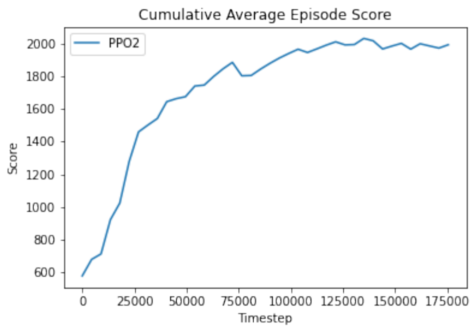
Policy	PPO2 Average Reward
Nature CNN	1992.5 ± 750.1
MLP	1943.3 ± 865.0

3.2.2 Sonic Experiments

With the listed adjustments, we used our MAML implementations to train a 2-layer MLP, and evaluated the fast learner for 100,000 or more timesteps. Below we compare the PPO2 baseline and CNN policy with the MAML agents.







As expected, given the previous baselines run with Reptile, the MAML approach performed significantly worse compared to PPO2. We suspect that a reason for this disparity is due to workarounds we implemented to address some high costs of MAML.

4 Conclusion

In this project we explored gradient-based meta-RL by experimenting with how different modifications affect MAML, and found that the choice of policy gradient in the meta-optimizer (as well as the number of gradient steps) had the largest affect on final performance. We also attempted to apply MAML to the *Sonic the Hedgehog* games in the OpenAI Retro Contest. However, our results were not as good as we had hoped. One reason for this was that we had to make several adjustments to the environment to stomach the large memory usage of MAML, which seem to have drastically hurt our performance. However, we also know that MAML in general was not a particularly successful approach during the contest. This could be because MAML usually aims to optimize for parameters that are able to reach good values for a new task in a very small (e.g. <3) number of gradient steps. However, the various Sonic levels are different enough and the game hard enough (even the best models could not get close to human performance) that trying to learn a good initialization to take a few gradient steps on unseen test levels is simply too ambitious.

References

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies, 2018.

Rein Houthooft, Richard Y. Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients, 2018.

Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl, 2018.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017a.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017b.