MAME : Model-Agnostic Meta-Exploration

Swaminathan Gurumurthy* **Sumit Kumar*** **Katia Sycara***

Abstract: Meta-Reinforcement learning approaches aim to develop learning procedures that can adapt quickly to a distribution of tasks with the help of a few examples. Developing efficient exploration strategies capable of finding the most useful samples becomes critical in such settings. Existing approaches towards finding efficient exploration strategies add auxiliary objectives to promote exploration by the pre-update policy, however, this makes the adaptation using a few gradient steps difficult as the pre-update (exploration) and post-update (exploitation) policies are often quite different. Instead, we propose to explicitly model a separate exploration policy for the task distribution. Having two different policies gives more flexibility in training the exploration policy and also makes adaptation to any specific task easier. We show that using self-supervised or supervised learning objectives for adaptation allows for more efficient inner-loop updates and also demonstrate the superior performance of our model compared to prior works in this domain.

Keywords: Meta-Learning, Exploration, Self-Supervised Learning

1 Introduction

Reinforcement learning (RL) approaches have seen many successes in recent years, from mastering the complex game of Go [1] to even discovering molecules [2]. However, a common limitation of these methods is their propensity to overfitting on a single task and inability to adapt to even slightly perturbed configuration [3]. On the other hand, humans have this astonishing ability to learn new tasks in a matter of minutes by using their prior knowledge and understanding of the underlying task mechanics. Drawing inspiration from human behaviors, researchers have proposed to incorporate multiple inductive biases and heuristics to help the models learn quickly and generalize to unseen scenarios. However, despite a lot of effort it has been difficult to approach human levels of data efficiency and generalization.

Meta reinforcement learning addresses these shortcomings by learning how to learn these inductive biases and heuristics from the data itself. It strives to learn an algorithm that allows an agent to succeed in a previously unseen task or environment when only limited experience is available. These inductive biases or heuristics can be induced in the model in various ways like optimization algorithm, policy, hyperparameters, network architecture, loss function, exploration strategies [4], *etc*. Recently, a class of parameter initialization based meta-learning approaches have gained attention like Model Agnostic Meta-Learning (MAML) [5]. MAML finds a good initialization for a model or a policy that can be adapted to a new task by fine-tuning with policy gradient updates from a few samples of that task.

Since the objective of meta-RL algorithms is to adapt to a new task from a few examples, efficient exploration strategies are crucial for quickly finding the optimal policy in a new environment. Some recent works have tried to address this problem by improving the credit assignment of the meta learning objective to the pre-update trajectory distribution [6, 7]. However, that requires transitioning the base policy from exploration behavior to exploitation behavior using one or few policy gradient updates. This limits the applicability of these methods to cases where the post-update (exploitation) policy is similar to the pre-update (exploration) policy and can be obtained with only a few updates. Additionally, for cases where pre- and post-update policies are expected to exhibit different behaviors, large gradient updates may result in training instabilities and poor performance at convergence.

*Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Corresponding authors: sgurumur@cs.cmu.edu, skumar2@cs.cmu.edu

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

To address this problem, we propose to explicitly model a separate exploration policy for the task distribution. The exploration policy is trained to find trajectories that can lead to fast adaptation of the exploitation policy on the given task. This formulation provides much more flexibility in training the exploration policy. We'll also show that separating the exploration and exploitation policy helps us improve/match the performance of the baseline approaches on meta-RL benchmark tasks. We further show that, in order to perform stable and fast adaptation to the new task, it is often more useful to use self-supervised or supervised learning approaches to perform the inner loop/meta updates, where possible. This also helps obtain more stable gradient update steps while training exploitation policy with the trajectories collected from a different exploration policy.

2 Related work

Meta-learning algorithms proposed in the RL community include approaches based on recurrent models [8, 9], metric learning [10, 11], and learning optimizers [12]. Recently, Finn et al. [5] proposed Model Agnostic Meta-Learning (MAML) which aims to learn a policy that can generalize to a distribution of tasks. Specifically, it aims to find a good initialization for a policy that can be adapted to any task sampled from the distribution by fine-tuning with policy gradient updates from a few samples of that task.

Efficient exploration strategies are crucial for finding trajectories that can lead to quick adaptation of the policy in a new environment. Recent works [13, 14] have proposed structured exploration strategies using latent variables to perform efficient exploration across successive episodes, however, they did not explicitly incentivize exploration in pre-update episodes. E-MAML [6] made the first attempt at assigning credit for the final expected returns to the pre-update distribution in order to incentivize exploration in each of the pre-update episodes. Rothfuss et al. [7] proposed Proximal Meta-Policy search (ProMP) where they incorporated the causal structure for more efficient credit assignment and proposed a low variance curvature surrogate objective to control the variance of the corresponding policy gradient update. However, these methods make use of a single base policy for both exploration and exploitation while relying on one or few gradient updates to transition from the exploration behavior to exploitation behavior. Over the next few sections, we illustrate that this approach is problematic and insufficient when the exploration and exploitation behaviors are quite different from each other.

A number of prior works have tried to utilize self-supervised objectives [15, 16, 17, 18, 19] to ease learning especially when the reward signal itself is insufficient to provide the required level of feedback. Drawing inspiration from these approaches, we modify the inner loop update/adaptation step in MAML using a self-supervised objective to allow more stable and faster updates. Concurrent to our work, Yang et al. [20] also decoupled exploration and adaptation policies where the latter is initialized as a learnable offset to the exploration policy.

3 Background

3.1 Meta-Reinforcement Learning

Unlike RL which tries to find an optimal policy for a single task, meta-RL aims to find a policy that can generalize to a distribution of tasks. Each task $\mathcal{T}$ sampled from the distribution $\rho(\mathcal{T})$ corresponds to a different Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \boldsymbol{P}, r, \gamma, H)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $\boldsymbol{P}$, reward function r, discount factor γ, and time horizon H. These MDPs are assumed to have similar state and action space but might differ in the reward function r or the environment dynamics $\boldsymbol{P}$. The goal of meta RL is to quickly adapt the policy to any task $\mathcal{T} \sim \rho(\mathcal{T})$ with the help of few examples from that task.

3.2 Credit Assignment in Meta-RL

MAML is a gradient-based meta-RL algorithm that tries to find a good initialization for a policy which can be adapted to any task $\mathcal{T} \sim \rho(\mathcal{T})$ by fine-tuning with one or more gradient updates using the sampled trajectories of that task. MAML maximizes the following objective function:

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\mathcal{T}\sim\rho(\mathcal{T})}\left[\mathbb{E}_{\boldsymbol{\tau}'\sim P_{\mathcal{T}}(\boldsymbol{\tau}'|\theta')}\left[R(\boldsymbol{\tau}')\right]\right] \\
\text{with} \quad \theta' &:= U(\theta, \mathcal{T}) = \theta + \alpha \nabla_\theta \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{T}}(\boldsymbol{\tau}|\theta)}\left[R(\boldsymbol{\tau})\right]
\end{aligned} \tag{1}
$$

where U is the update function that performs one policy gradient ascent step to maximize the expected reward $R(\boldsymbol{\tau})$ obtained on the trajectories $\boldsymbol{\tau}$ sampled from task $\mathcal{T}$.

Rothfuss et al. [7] showed that the gradient of the objective function $J(\theta)$ in Eq. 1 can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathcal{T}\sim\rho(\mathcal{T})}\left[\mathbb{E}_{\boldsymbol{\tau}\sim P_\mathcal{T}(\boldsymbol{\tau}|\theta)\boldsymbol{\tau}'\sim P_\mathcal{T}(\boldsymbol{\tau}'|\theta')}\left[\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}') + \nabla_\theta J_{\text{pre}}\left(\boldsymbol{\tau},\boldsymbol{\tau}'\right)\right]\right]$$

where,

$$\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}') = \underbrace{\nabla_{\theta'}\log\pi_\theta(\boldsymbol{\tau}')R(\boldsymbol{\tau}')}_{\nabla_{\theta'}J^{\text{outer}}}\underbrace{\left(I + \alpha R(\boldsymbol{\tau})\nabla^2_\theta\log\pi_\theta(\boldsymbol{\tau})\right)}_{\text{transformation from }\theta'\text{ to }\theta} \tag{2}$$

$$\nabla_\theta J_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}') = \alpha\nabla_\theta\log\pi_\theta(\boldsymbol{\tau})\left(\underbrace{\left(\nabla_\theta\log\pi_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})\right)^\top}_{\nabla_\theta J^{\text{inner}}}\underbrace{\left(\nabla_{\theta'}\log\pi_{\theta'}(\boldsymbol{\tau}')R(\boldsymbol{\tau}')\right)}_{\nabla_{\theta'}J^{\text{outer}}}\right) \tag{3}$$

The first term $\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}')$ corresponds to a policy gradient step on the post-update policy $\pi_{\theta'}$ w.r.t.. the post-update parameters θ' which is then followed by a linear transformation from θ' to θ (pre-update parameters). Note that $\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}')$ optimizes θ to increase the likelihood of the trajectories $\boldsymbol{\tau}'$ that lead to higher returns given some trajectories $\boldsymbol{\tau}$. However, this term does not optimize θ to yield trajectories $\boldsymbol{\tau}$ that lead to good adaptation steps. That is, infact, done by the second term $\nabla_\theta J_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}')$. It optimizes for the pre-update trajectory distribution, $P_\mathcal{T}(\boldsymbol{\tau}|\theta)$, i.e, increases the likelihood of trajectories $\boldsymbol{\tau}$ that lead to good adaptation steps.

During optimization, MAML only considers $J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}')$ and ignores $J_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}')$. Thus MAML finds a policy that adapts quickly to a task given relevant experiences, however, the policy is not optimized to gather useful experiences from the environment that can lead to fast adaptation.

Rothfuss et al. [7] proposed ProMP where they analyze this issue with MAML and incorporate $\nabla_\theta J_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}')$ term in the update as well. They used The Infinitely Differentiable Monte-Carlo Estimator (DICE) [21] to allow causal credit assignment on the pre-update trajectory distribution, however, the gradients computed by DICE still have high variance. To remedy this, they proposed a low variance (and slightly biased) approximation of the DICE based loss that leads to stable updates.

4 Method

The pre-update and post-update policies are often expected to exhibit very different behaviors, i.e, exploration and exploitation behaviors respectively. For instance, consider a 2D environment where a task corresponds to reaching a goal location sampled randomly from a semi-circular region (example shown in appendix). The agent receives a reward only if it lies in some vicinity of the goal location. The optimal pre-update or exploration policy is to move around in the semi-circular region whereas the ideal post-update or exploitation policy will be to reach the goal state as fast as possible once the goal region is discovered. Clearly, the two policies are expected to behave very differently. In such cases, transitioning a single policy from pure exploration phase to pure exploitation phase via policy gradient updates will require multiple steps. Unfortunately, this significantly increases the computational and memory complexities of the algorithm. Furthermore, it may not even be possible to achieve this transition via few gradient updates. This raises an important question: DO WE REALLY NEED TO USE THE PRE-UPDATE POLICY FOR EXPLORATION AS WELL? CAN WE USE A SEPARATE POLICY FOR EXPLORATION?

Using separate policies for pre-update and post-update sampling: The straightforward solution to the above problem is to use a separate exploration policy μ_ϕ responsible for collecting trajectories for the inner loop updates to get θ'. Following that, the post-update policy $\pi_{\theta'}$ can be used to collect trajectories for performing the outer loop updates. Unfortunately, this is not as simple as it sounds. To understand this, let's look at the inner loop updates:

$$U(\theta,\mathcal{T}) = \theta + \alpha\nabla_\theta\mathbb{E}_{\boldsymbol{\tau}\sim P_\mathcal{T}(\boldsymbol{\tau}|\theta)}\left[R(\boldsymbol{\tau})\right]$$

When the exploration policy is used for sampling trajectories, we need to perform importance sampling. The update would thus become:

$$U(\theta,\mathcal{T}) = \theta + \alpha\nabla_\theta\mathbb{E}_{\boldsymbol{\tau}\sim Q_\mathcal{T}(\boldsymbol{\tau}|\phi)}\left[\frac{P_\mathcal{T}(\boldsymbol{\tau}|\theta)}{Q_\mathcal{T}(\boldsymbol{\tau}|\phi)}R(\boldsymbol{\tau})\right]$$

where $P_\mathcal{T}(\boldsymbol{\tau}|\theta)$ and $Q_\mathcal{T}(\boldsymbol{\tau}|\phi)$ represent the trajectory distribution sampled by π_θ and μ_ϕ respectively. Note that the above update is an off-policy update which results in high variance estimates when the

two trajectory distributions are quite different from each other. This makes it infeasible to use the importance sampling update in the current form. In fact, this is a more general problem that arises even in the on-policy regime. The policy gradient updates in the inner loop results in both $\nabla_\theta J_{\text{pre}}$ and $\nabla_\theta J_{\text{post}}$ terms being high variance. This stems from the mis-alignment of the outer gradients $(\nabla_{\theta'} J^{outer})$ and the inner gradient, hessian $\left(\nabla_\theta J^{\text{inner}}, \nabla^2_\theta \log \pi_\theta(\boldsymbol{\tau})\right)$ terms appearing in Eq. 2 and 3. This motivates our second question: DO WE REALLY NEED THE PRE-UPDATE GRADIENTS TO BE POLICY GRADIENTS? CAN WE USE A DIFFERENT OBJECTIVE IN THE INNER LOOP TO GET MORE STABLE UPDATES?

Using a self-supervised/supervised objective for the inner loop update step: The instability in the inner loop updates arises due to the high variance nature of the policy gradient update. Note that the objective of inner loop update is to provide some task specific information to the agent with the help of which it can adapt its behavior in the new environment. We believe that this could be achieved using some form of self-supervised or supervised learning objective in place of policy gradient in the inner loop to ensure that the updates are more stable. We propose to use a network for predicting some task (or MDP) specific property like reward function, expected return or value. During the inner loop update, the network updates its parameters by minimizing its prediction error on the given task. Unlike prior meta-RL works where the task adaptation in the inner loop is done by policy gradient updates, here, we update some parameters shared with the exploitation policy using a supervised loss objective function which leads to stable updates during the adaptation phase. However, note that the variance and usefulness of the update depends heavily on the choice of the self-supervision/supervision objective. We delve into this in more detail in Section 4.1.1.

4.1 Model

Our proposed model comprises of three modules, the exploration policy $\mu_\phi(s)$, the exploitation policy $\pi_{\theta,z}(s)$, and the self-supervision network $M_{\beta,z}(s,a)$. Note that $M_{\beta,z}$ and $\pi_{\theta,z}$ share a set of parameters z while containing their own set of parameters β and θ respectively. We describe our proposed model in Fig. 1. Our model differs from E-MAML/ProMP because of the separate exploration policy, the separation of task-specific parameters z and task agnostic parameters θ, and the self-supervised update as shown in Fig. 1.

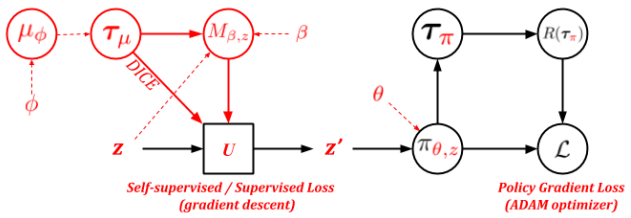


Figure 1: Model Flowchart: Black structures are those consistent with E-MAML/ProMP. Red structures are the key differences with E-MAML/ProMP (The thin-dotted arrow means the parameters related to that node.). Specifically, the pre-update trajectories $\boldsymbol{\tau}$ are now collected using a separate exploration policy μ_ϕ. The corresponding adaptation update is performed using a self-supervised/supervised objective, $\left(M_{\beta,z}(s,a) - \overline{M}(s,a)\right)^2$, on z to give z' and the policy $\pi_{\theta,z'}$ is parameterized using the task specific parameters z' and the task agnostic parameters θ

The agent first collects a set of trajectories $\boldsymbol{\tau}$ using its exploration policy μ_ϕ for each task $\mathcal{T} \sim \rho(\mathcal{T})$. It then updates the shared parameter z by minimizing the regression loss $\left(M_{\beta,z}(s,a) - \overline{M}(s,a)\right)^2$ on the sampled trajectories $\boldsymbol{\tau}$:

$$z' = U(z, \mathcal{T}) = z - \alpha \nabla_z \mathbb{E}_{\boldsymbol{\tau} \sim Q_\mathcal{T}(\boldsymbol{\tau}|\phi)} \left[\sum_{t=0}^{H-1} \left(M_{\beta,z}(s_t,a_t) - \overline{M}(s_t,a_t)\right)^2 \right] \tag{4}$$

where, $\overline{M}(s,a)$ is the target, which can be any task specific quantity like reward, return, value, next state etc. After obtaining the updated parameters z' for each task $\mathcal{T}$, the agent samples the (validation) trajectories $\boldsymbol{\tau}'$ using its updated exploitation policy $\pi_{\theta,z'}$. Effectively, z' encodes the necessary information regarding the task that helps an agent in adapting its behavior to maximize its expected return whereas θ remain task agnostic. A similar approach was proposed by Zintgraf et al. [22] to learn task-specific behavior using context variable with MAML.

The collected trajectories are then used to perform a policy gradient update to all parameters z, θ, ϕ and β using the following objective:

$$J(z', \theta) = \mathbb{E}_{\mathcal{T} \sim \rho(\mathcal{T})} \left[\mathbb{E}_{\boldsymbol{\tau}_\pi^\mathcal{T} \sim P_\mathcal{T}(\boldsymbol{\tau}_\pi^\mathcal{T}|\theta,z')} \left[R(\boldsymbol{\tau}_\pi^\mathcal{T}) \right] \right] \tag{5}$$

In order to allow multiple outer-loop updates, we use the PPO [23] objective instead of the vanilla policy gradient objective to maximize Eq. 5. Furthermore, we don't perform any outer loop updates on z and treat it as a shared latent variable with fixed initial values of 0 as proposed in [22]. The reason being, that the bias term in the layers connecting z to the respective networks would learn to compensate for the initialization. We only update z to z' in the inner loop to obtain a task specific latent variable.

Note that in all the prior meta reinforcement learning algorithms, both the inner-loop update and the outer-loop update are policy gradient updates. In contrast, in this work, the inner-loop update is a supervised learning gradient descent update whereas the outer-loop update remains a policy gradient update.

The outer loop gradients w.r.t. ϕ, $\nabla_\phi J(z', \theta)$ can be simplified by multiplying the DICE [21] operator inside the expectation in Eq. 4 as proposed by Rothfuss et al. [7]. This allows the gradients w.r.t. ϕ to be computed in a straightforward manner with back-propagation. This also eliminates the need to apply the policy gradient trick to expand Eq. 4 for gradient computation. The inner loop update then becomes:

$$z' = U(z, \mathcal{T}) = z - \alpha \nabla_z \mathbb{E}_{\boldsymbol{\tau} \sim Q_\mathcal{T}(\boldsymbol{\tau}|\phi)} \left[\sum_{t=0}^{H-1} \left(\prod_{t'=0}^{t} \frac{\mu_\phi(a_{t'}|s_{t'})}{\perp(\mu_\phi(a_{t'}|s_{t'}))} \right) \left(M_{\beta,z}(s_t, a_t) - \overline{M}(s_t, a_t) \right)^2 \right]$$

where $\perp$ is the stop gradient operator as introduced in [21].

The pseudo-code of our algorithm is shown in appendix (see algorithm 7.1). However, we found that implementing algorithm 7.1 as it is, using DICE leads to high variance gradients for ϕ, resulting in instability during training and poor performance of the learned model. To understand this, let's look at the vanilla DICE gradients for the exploration parameters ϕ, which can be written as follows:

$$\nabla_\phi J(z', \theta) = \mathbb{E}_{\mathcal{T} \sim \rho(\mathcal{T})} \Bigg[\mathbb{E}_{\boldsymbol{\tau} \sim Q_\mathcal{T}(\boldsymbol{\tau}|\phi)} \sum_{t=0}^{H-1} \alpha \nabla_\phi \log \mu_\phi(s_t) \Bigg[\sum_{t'=t}^{H-1} \left(\mathbb{E}_{\boldsymbol{\tau}' \sim P_\mathcal{T}(\boldsymbol{\tau}'|\theta, z')} \left(\nabla_{z'} \log \pi_{\theta, z'}(\boldsymbol{\tau}') R(\boldsymbol{\tau}') \right)^\top \right) \left(\nabla_z \left(M_{\beta,z}(s_t, a_t) - \overline{M}(s_t, a_t) \right)^2 \right) \Bigg] \Bigg]$$

The above expression can be viewed as a policy gradient update:

$$\nabla_\phi J(z', \theta) = \mathbb{E}_{\mathcal{T} \sim \rho(\mathcal{T})} \left[\mathbb{E}_{\boldsymbol{\tau} \sim Q_\mathcal{T}(\boldsymbol{\tau}|\phi)} \sum_{t=0}^{H-1} \alpha \nabla_\phi \log \mu_\phi(s_t) R_t^\mu \right] \tag{6}$$

with returns

$$R_t^\mu = \left[\sum_{t'=t}^{H-1} \left(\mathbb{E}_{\boldsymbol{\tau}' \sim P_\mathcal{T}(\boldsymbol{\tau}'|\theta, z')} \left(\nabla_{z'} \log \pi_{\theta, z'}(\boldsymbol{\tau}') R(\boldsymbol{\tau}') \right)^\top \right) \left(\nabla_z \left(M_{\beta,z}(s_t, a_t) - \overline{M}(s_t, a_t) \right)^2 \right) \right] \tag{7}$$

Note that the variance depends on the policy gradient terms computed in the outer-loop and the choice of self-supervision. We'll explain the latter in Sec 4.1.1. However, irrespective of the choice, we can use value function based variance reduction ([24]) by substituting the above computed returns with advantages, i.e, we replace the return R_t^μ in Eq. 7 with an advantage estimate A_t^μ and use a PPO ([23]) objective to allow multiple outer loop updates: :

$$\nabla_\phi \hat{J}(z', \theta) = \mathbb{E}_{\mathcal{T} \sim \rho(\mathcal{T})} \left[\mathbb{E}_{\boldsymbol{\tau} \sim Q_\mathcal{T}(\boldsymbol{\tau}|\phi_o)} \left[\sum_{t=0}^{H-1} \alpha \nabla_\phi \min \left(\frac{\mu_\phi(s_t)}{\mu_{\phi_o}(s_t)} A_t^\mu \, , \, \text{clip}_{1-\epsilon}^{1+\epsilon} \left(\frac{\mu_\phi(s_t)}{\mu_{\phi_o}(s_t)} \right) A_t^\mu \right) \right] \right]$$

where,

$$A_t^\mu = r_t^\mu + V_{t+1}^\mu - V_t^\mu \tag{8}$$

where V_t^μ is computed using a neural network or a linear feature baseline [25] fitted on the returns R_t^μ. where r_t^μ is given by:

$$r_t^\mu = \left(\mathbb{E}_{\boldsymbol{\tau}' \sim P_\mathcal{T}(\boldsymbol{\tau}'|\theta, z')} \left(\nabla_{z'} \log \pi_{\theta, z'}(\boldsymbol{\tau}') R(\boldsymbol{\tau}') \right)^\top \right) \left(\nabla_z \left(M_{\beta,z}(s_t, a_t) - \overline{M}(s_t, a_t) \right)^2 \right) \tag{9}$$

4.1.1 Self-Supervised/Supervised Objective

It is important to note that the self-supervised/supervised learning objective not only guides the adaptation step but also influences the exploration policy update as seen in Eq. 6 and 7. We mentioned above that the self-supervised/supervised learning objective could be as simple as a value/reward/return/next state prediction for each state (state-action pair). However, the exact choice of the objective can be critical to the final performance and stability. From the perspective of the adaptation step, the only criterion is that the self-supervised objective should contain enough task specific information to allow a useful adaptation step. For example, it would not be a good idea to use the rewards self-supervision in sparse/noisy reward scenarios or the next state predictions as self-supervision when the dynamics model does not change much over tasks since the self-supervision updates in such cases will not carry enough task specific information. From the perspective of the exploration policy updates, an additional requirement would be to ensure that the returns computed in Eq. 7 are low variance and unbiased, which further translates to saying $\nabla_z \left(M_{\beta,z}(s_t, a_t) - \overline{M}(s_t, a_t)\right)^2$ should ideally be low variance and unbiased. For example, using the cumulative future returns as self-supervision might lead to a very high variance update in certain environments. Thus, finding a generalizable self-supervision/supervision objective which satisfies both properties mentioned above across all scenarios is a challenging task.

In our experiments, we found that, using N-step return prediction for supervision works reasonably well across all the environments. This acts as a trade-off between predicting the full return (which was high variance but also more task-specific info) and the reward (which was lower variance but lower task-specific info). Hence, $\overline{M}(s_t, a_t)$ becomes $\overline{M}(s_t, a_t, s_{t+1}, a_{t+1}, ...s_{t+N-1}, a_{t+N-1}) = \sum_{t'=t}^{t+N-1} r(s'_t, a'_t)$. However, using $M_{\beta,z}$ to directly predict $\overline{M}$ would still lead to high variance in $\nabla_z \left(M_{\beta,z}(s_t, a_t) - \overline{M}\right)^2$. Thus, we use the truncated N-step successor representations [26] (similar to N-step returns) $m_\beta(s_t, a_t)$ and a linear layer on top of that to compute $M_{\beta,z}(s_t, a_t) = w_\beta^T m_\beta(s_t, a_t)$. Using the successor representations can effectively be seen as using a more accurate/powerful baseline than directly predicting the N-step returns using the (s_t, a_t) pair.

5 Experiments

We evaluate our proposed model on a set of 6 benchmark continuous control environments, `Ant-Fwd-Bwd`, `Half-Cheetah-Fwd-Bwd`, `Half-Cheetah-Vel`, `Walker2D-Fwd-Bwd`, `Walker2D-Rand-Params` and `Hopper-Rand-Params` used in [7]. We also compare our method with 3 baseline approaches: MAML, EMAML and ProMP. Furthermore, we also perform ablation experiments to analyze different components and design choices of our model on a toy 2D point environment proposed by [7].

The details of the network architecture and the hyperparameters used for learning have been mentioned in the appendix. We would like to state that we have not performed much hyperparameter tuning due to computational constraints and we expect the results of our method to show further improvements with further tuning. Also, we restrict ourselves to a single adaptation step in all environments for the baselines as well as our method, but it can be easily extended to multiple gradient steps as well by conditioning the exploration policy on the latent parameters z.

The results of the baselines for the benchmark environments have been borrowed directly from the the official ProMP website [2]. For the point environments, we have used their official implementation[3].

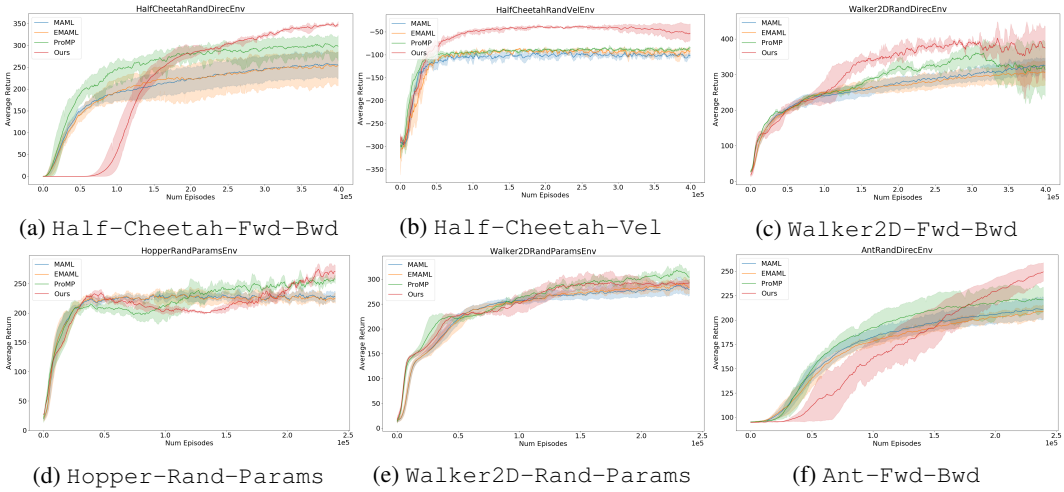
5.1 Meta RL Benchmark Continuous Control Tasks.

The continuous control tasks require adaptation either across reward functions (`Ant-Fwd-Bwd`, `Half-Cheetah-Fwd-Bwd`, `Half-Cheetah-Vel`, `Walker2D-Fwd-Bwd`) or across dynamics (`Walker2D-Rand-Params` and `Hopper-Rand-Params`). We set the horizon length to be 100 in `Ant-Fwd-Bwd` and `Half-Cheetah-Fwd-Bwd` environments and 200 in others in accordance with the practice in [7]. The performance plots for all the 4 algorithms are shown in Fig. 2. In all the environments, our proposed method outperforms or achieves similar performance to other method in terms of asymptotic performance.

Our algorithm performs particularly well in `Half-Cheetah-Fwd-Bwd`, `Half-Cheetah-Vel`, `Walker2D-Fwd-Bwd` and `Ant-Fwd-Bwd` environments where the N-step returns are informa-

[2] https://sites.google.com/view/pro-mp/experiments

[3] https://github.com/jonasrothfuss/ProMP



(a) `Half-Cheetah-Fwd-Bwd` (b) `Half-Cheetah-Vel` (c) `Walker2D-Fwd-Bwd`

(d) `Hopper-Rand-Params` (e) `Walker2D-Rand-Params` (f) `Ant-Fwd-Bwd`

Figure 2: Comparison of our method with 3 baseline methods on the Meta-RL Benchmark tasks.

tive. In `Ant-Fwd-Bwd` and `Half-Cheetah-Fwd-Bwd` environments, although we reach similar asymptotic performance as ProMP, the convergence is slower in the initial stages of training. This is because training multiple networks together can make training slower especially in the initial stages of training especially when the training signal isn't strong enough. Note that in `Walker2D-Rand-Params` and `Hopper-Rand-Params` environments, although our method converges as well as the baselines, it doesn't do much better in terms of peak performance. This could be attributed to the selection of the self-supervision signal. A more appropriate self-supervision signal for these environments would be the next state or successor state prediction since the task distribution in these environments corresponds to the variation in model dynamics and not just reward function. This shows that the choice of the self-supervision signal plays an important role in the model's performance. To further understand these design choices we perform some ablations on a toy environment in section 5.2.1.

5.2 2D Point Navigation.

We show the performance plots for ProMP, MAML-TRPO, MAML-VPG and our algorithm in the sparse reward `2DPointEnvCorner` environment (proposed in [7]) in Fig. 2. Each task in this environment corresponds to reaching a randomly sampled goal location (one of the four corners) in a 2D environment. This is a sparse reward task where the agent receives a reward only if it is sufficiently close to the goal location. In this environment, the agent needs to perform efficient exploration and use the sparse reward trajectories to perform stable updates, both of which are salient aspects of our algorithm. Our method is able to achieve this and reaches peak performance while showing stable behavior. ProMP, on the other hand, also reaches the peak performance but shows more unstable behavior than in the dense reward scenarios, although it manages to reach similar peak performance to our method. The other baselines struggle to do much in this environment since they do not explicitly incentivize exploration for the pre-update policy.

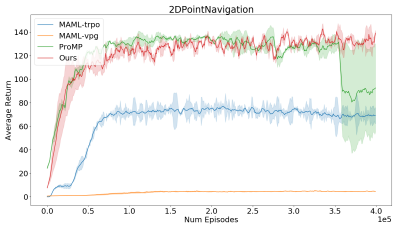


Figure 3: 2D Point Navigation

5.2.1 Ablation Study

We perform several ablation experiments to analyze the impact of different components of our algorithm on 2D point navigation task. Fig. 4 shows the performance plots for the following 5 different variants of our algorithm:

VPG-Inner loop: The semi-supervised/supervised loss in the inner loop is replaced with the vanilla policy gradient loss as in MAML while using the exploration policy to sample the pre-update trajectories. This variant illustrates our claim

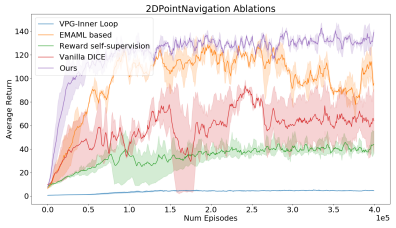


Figure 4: Ablation results

of unstable inner loop updates when naively using an exploration policy. As expected, this model performs poorly due to the high variance off-policy updates in the inner loop.

Reward Self-Supervision : A reward based self-supervised objective is used instead of return based self-supervision, i.e, the self-supervision network M now predicts the reward instead of the N-step return at each time step. This variant is stable but struggles to reach peak performance since the task is sparse reward. This shows that the choice of self-supervision objective is also important and needs to be chosen carefully.

Vanilla DiCE: In this variant, we directly use the DICE gradients to perform updates on ϕ instead of using the low variance gradient estimator. The leads to higher variance updates and unstable training as can be seen from the plots. This shows that the low variance gradient estimate has a major contribution to the stability during training.

E-MAML Based : Here, we used an E-MAML [6] type objective to compute the gradients w.r.t. ϕ instead of using DICE, i.e, directly used policy gradient updates on μ_ϕ but instead with returns computed on post-update trajectories. This variant ignores the causal credit assignment from output to inputs. Thus, the updates are of higher variance, leading to more unstable updates, although it manages to reach good performance.

Ours : The low variance estimate of the DICE gradients is used to compute updates for ϕ along with N-step return based self-supervision for inner loop updates. Our model reaches peak performance and exhibits stable training due to low variance updates.

6 Discussions and Conclusion

Unlike conventional meta-RL approaches, we proposed to explicitly model a separate exploration policy for the task distribution. Having two different policies gives more flexibility in training the exploration policy and also makes adaptation to any specific task easier. The above experiments illustrate that our approach provides more stable updates and better asymptotic performance as compared to ProMP when the pre-update and post-update policies are very different. Even when that is not the case, our approach matches or surpasses the baselines in terms of asymptotic performance. More importantly, this shows that in most of these tasks, separating the exploration and exploitation policies can yield better performance if properly done. From our ablation studies, we show that the self-supervised objective plays a huge role in improving stability of the updates and the choice of the self-supervised objective can be critical in some cases (e.g, predicting reward v/s return). Further, we also show through the above experiments that the variance reduction techniques used in the objective of exploration policy is important for achieving stable behavior. However, we would like to emphasize that the idea of using a separate exploration and exploitation policy is much more general and doesn't need to be restricted to MAML. Given the requirements of sample efficiency of the adaptation steps in the meta-learning setting, exploration is a very crucial ingredient and has been vastly under explored. Thus, we would like to explore the following extensions as future work:

- Explore other techniques of self-supervision that can be more generally used across environments and tasks.
- Decoupling the exploration and exploitation policies allows us to perform off-policy updates. Thus, we plan to test it as a natural extension of our approach.
- Explore the use of having separate exploration and exploitation policies in other meta-learning approaches.

Acknowledgments

This work has been funded by AFOSR award FA9550-15-1-0442 and AFOSR/AFRL award FA9550-18-1-0251. We would like to thank Tristan Deleu, Maruan Al-Shedivat, Anirudh Goyal and Lisa Lee for their insightful and fruitful discussions and Tristan Deleu, Jonas Rothfuss and Dennis Lee for opensourcing the repositories and result files.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550 (7676):354, 2017.

[2] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):48, 2017.

[3] C. Zhang, O. Vinyals, R. Munos, and S. Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[4] A. Sharaf and H. Daumé III. Meta-learning for contextual bandit exploration. *arXiv preprint arXiv:1901.08159*, 2019.

[5] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[6] 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. *arXiv preprint arXiv:1803.01118*, 2018.

[7] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel. Promp: Proximal meta-policy search. *arXiv preprint arXiv:1810.06784*, 2018.

[8] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl ^{2}: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[9] C. Finn and S. Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *arXiv preprint arXiv:1710.11622*, 2017.

[10] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[11] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[12] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. URL http://arxiv.org/abs/1803.02999.

[13] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pages 5307–5316, 2018.

[14] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint arXiv:1903.08254*, 2019.

[15] P. Mahmoudieh. Self-supervision for reinforcement learning. 2017.

[16] C. Florensa, J. Degrave, N. Heess, J. T. Springenberg, and M. Riedmiller. Self-supervised learning of image embedding for continuous control. *arXiv preprint arXiv:1901.00943*, 2019.

[17] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[18] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[19] M. Wortsman, K. Ehsani, M. Rastegari, A. Farhadi, and R. Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. *arXiv preprint arXiv:1812.00971*, 2018.

[20] Y. Yang, K. Caluwaerts, A. Iscen, J. Tan, and C. Finn. Norml: No-reward meta learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 323–331. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[21] J. Foerster, G. Farquhar, M. Al-Shedivat, T. Rocktäschel, E. P. Xing, and S. Whiteson. Dice: The infinitely differentiable monte-carlo estimator. *arXiv preprint arXiv:1802.05098*, 2018.

[22] L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson. Caml: Fast context adaptation via meta-learning. *arXiv preprint arXiv:1810.03642*, 2018.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[25] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[26] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

7 Appendix

7.1 Algorithm

Require: Task distribution $\rho(\mathcal{T})$, step sizes α, η
while not converge **do**
  Sample a batch of tasks $\mathcal{T}_i \sim \rho(\mathcal{T})$
  for all sampled tasks $\mathcal{T}_i$ **do**
    Collect pre-update trajectories $\boldsymbol{\tau}_{\mu}^{\mathcal{T}_i}$ using $\mu_\phi(s)$
    Update z by minimizing $\left(M_{\beta,z}(s,a) - \overline{M}(s,a)\right)^2$:

$$z' = U(z, \mathcal{T}) = z - \alpha \nabla_z \mathbb{E}_{\boldsymbol{\tau}_{\mu}^{\mathcal{T}_i} \sim Q_{\mathcal{T}_i}(\boldsymbol{\tau}|\phi)} \left[\sum_{t=0}^{H-1} \left(\prod_{t'=0}^{t} \frac{\mu_\phi(a_{t'}|s_{t'})}{\perp\left(\mu_\phi(a_{t'}|s_{t'})\right)} \right) \left(M_{\beta,z}(s,a) - \overline{M}(s,a)\right)^2 \right]$$

    where $\left(\prod_{t'=0}^{t} \frac{\mu_\phi(a_{t'}|s_{t'})}{\perp(\mu_\phi(a_{t'}|s_{t'}))} \right)$ is the DiCE operator
    Collect post-update trajectory $\boldsymbol{\tau}_{\pi}^{\mathcal{T}_i}$ using $\pi_{\theta,z'}(s)$
  Policy gradient update w.r.t. post-update trajectory $\boldsymbol{\tau}_{\pi}^{\mathcal{T}_i}$ to optimize all parameters z, θ, ϕ, β, with the objective

$$J(z', \theta) = \mathbb{E}_{\mathcal{T} \sim \rho(\mathcal{T})} \left[\mathbb{E}_{\boldsymbol{\tau}_{\pi}^{\mathcal{T}} \sim P_{\mathcal{T}}(\boldsymbol{\tau}_{\pi}^{\mathcal{T}}|\theta,z')} \left[R(\boldsymbol{\tau}_{\pi}^{\mathcal{T}}) \right] \right]$$

7.2 Additional experiments

7.2.1 SemiCircleEnvironment

We perform some additional experiments on another toy environment to illustrate the exploration behavior shown by our model and demonstrate the benefits of using different exploration and exploitation policies. Fig 5 shows an environment where the agent is initialized at the center of the semi-circle. Each task in this environment corresponds to reaching a goal location (red dot) randomly sampled from the semi circle (green dots). This is also a sparse reward task where the agent receives a reward only if it is sufficiently close to the goal location. However, unlike the previous environments, we only allow the agent to sample 2 pre-update trajectories per task in order to identify the goal location. Thus the agent has to explore efficiently at each exploration step in order to perform reasonably at the task. Fig 5 shows the trajectories taken by our exploration agent (orange and blue) and the exploitation/trained agent (green). Clearly, our agent has learnt to explore the environment. However, we know that a policy going around the periphery of the semi-circle would be a more useful exploration policy. In this environment we know that this exploration behavior can be reached by simply maximizing the environment rewards collected by the exploration policy. Fig. 7 shows this experiment where the exploration policy is trained using environment reward maximization while everything else is kept unchanged. We call this variant Ours-EnvReward. We also show the trajectories traversed by promp in Fig 6. It is clear that it struggles to learn different exploration and exploitation behaviors. Fig. 8 shows the performance of our two variants along with the baselines. This experiment shows that decoupling the exploration and exploitation policies also allows us, the designers more flexibility at training them, i.e, it allows us to add any domain knowledge we might have regarding the exploration or the exploitation policies to further improve the performance.

7.2.2 Varying number of adaptation trajectories collected

We additionally wanted to test the sensitivity of the algorithms to the number of trajectories collected in the inner loop. This is crucial because at test time, the algorithms would only be collecting trajectories for the inner loop update, i.e, for adaptation. We test this on the HalfCheetah-Vel Environment with varying numbers of inner loop adaptation trajectories namely, 2, 5, 10 and 20. However to keep the updates stable, we increase the meta batch size (number of tasks sampled for each update) proportionally to 400, 160, 80 and 40 respectively. Figure 11 shows the plots for these variants for ProMP and our model. We notice that the performance of our model stays roughly constant across varying values of the number of adaptation trajectories whereas ProMP shows degradation in performance as the number of adaptation trajectories decrease. This shows that each of the trajectories we sample performs efficient exploration. Note that the last pair with (20,40) correspond to the standard settings of hyper-parameters which we (and other papers before us) have used for the above experiments.

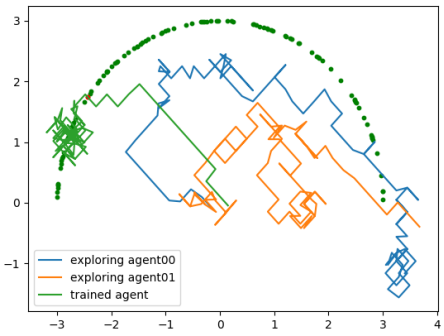


Figure 5: Ours

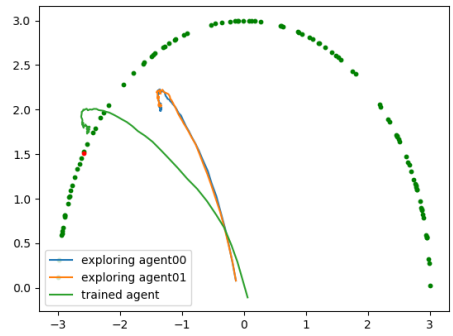


Figure 6: ProMP

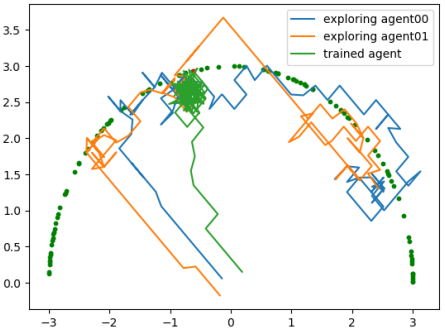


Figure 7: Ours-EnvReward : μ_ϕ maximizes its environment rewards

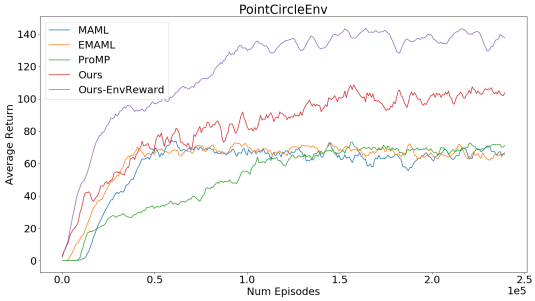


Figure 8: Comparison with baselines

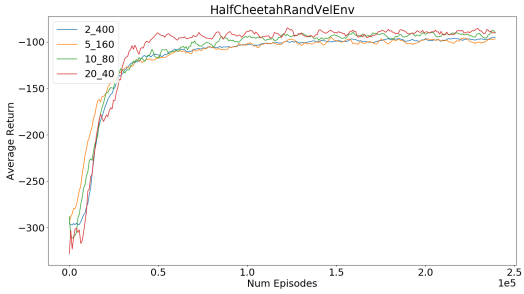


Figure 9: ProMP

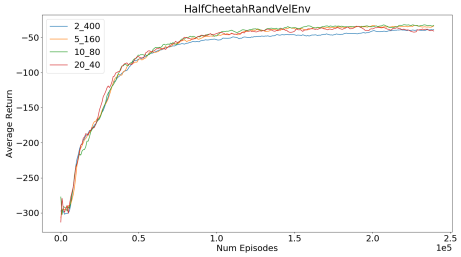


Figure 10: Ours

Figure 11: Comparison with varying numbers of adaptation trajectories. In the legend "x_y" corresponds to a run with x adaptation trajectories and y tasks sampled for each updated.

7.3 Hyper-parameters and Details

For all the experiments, we treat the shared parameter z as a latent embedding with a fixed initial value of $\bar{0}$. The exploitation policy $\pi_{\theta,z}(s)$ and the self-supervision network $M_{\beta,z}(s,a)$ concatenates z with their respective inputs. All the three networks (π, μ, M) have the same architecture (except inputs and output sizes) as that of the policy network in [7] for all experiments. We also stick to the same values of hyper-parameters such as inner loop learning rate, gamma, tau and number of outer loop updates. We keep a constant embedding size of 32 and a constant N=15 (for computing the N-step returns) across all experiments and runs. We use the Adam [27] optimizer with a learning rate of $7e-4$ for all parameters. Also, we restrict ourselves to a single adaptation step in all environments, but it can be easily extended to multiple gradient steps as well by conditioning the exploration policy on the latent parameters z.