Learning Exploration Policies for Model-Agnostic Meta-Reinforcement Learning

Swaminathan Gurumurthy [1] **Sumit Kumar** [1] **Katia Sycara** [1]

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 to 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 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 stabilizes the training process and further demonstrate the superior performance of our model compared to prior works in this domain.

1. Introduction

Reinforcement learning (RL) approaches have seen many successes in recent years, from mastering the complex game of Go (Silver et al., 2017) to even discovering molecules (Olivecrona et al., 2017). 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 (Zhang et al., 2018). 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-RL tries to address these shortcomings by learning these inductive biases and heuristics from the data itself. These inductive biases or heuristics can be induced in the model in various ways like optimization, policy initialization, loss function, exploration strategies, etc. Recently, a class of policy initialization based meta-learning approaches have gained attention like Model Agnostic Meta-Learning (MAML) (Finn et al., 2017). MAML finds a good initialization for a policy that can be adapted to a new task by fine-tuning with policy gradient updates from a few samples of that task.

Given the objective of meta RL algorithms 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 (Gupta et al., 2018a;b) have tried to address this problem by using latent variables to model the distribution of exploration behaviors. Another set of approaches (Stadie et al., 2018; Rothfuss et al., 2018) focus on improving the credit assignment of the meta learning objective to the pre-update trajectory distribution. However, all these prior works use one or few policy gradient updates to transition from pre- to post-update policy. 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. Also, for cases where pre- and post-update policies are expected to exhibit different behaviors, large gradient updates may result in training instabilities and lack of convergence. To address this problem, we propose to explicitly model a separate exploration policy for the distribution of tasks. 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. In the process, we also establish that, in order to adapt as quickly as possible to the new task, it is often more useful to use self-supervised or supervised learning approaches, where possible, to get more effective updates.

[1]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

2. Background

2.1. Meta-Reinforcement Learning

Unlike RL which tries to find an optimal policy for a single task, meta-RL aims to learn 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). These MDPs have similar state and action space but might differ in the reward function or the environment dynamics. 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.

2.2. Credit Assignment in Meta-RL

Finn et al. (2017) introduced MAML - 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:

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

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. (2018) showed that the gradient of the objective function $J(\theta)$ can be written as:

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

$$\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}') = \underbrace{\nabla_{\theta'}\log\pi_\theta(\boldsymbol{\tau}')R(\boldsymbol{\tau}')}_{\nabla_{\theta'}J^{\text{outer}}} \quad (2)$$

$$\underbrace{\left(I + \alpha R(\boldsymbol{\tau})\nabla^2_\theta \log\pi_{\theta'}(\boldsymbol{\tau})\right)}_{\text{transformation from } \theta' \text{ to } \theta} \quad (3)$$

$$\nabla_\theta J_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}') = \alpha\nabla_\theta\log\pi_\theta(\boldsymbol{\tau})\Bigg(\underbrace{\left(\nabla_\theta\log\pi_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})\right)^\top}_{\nabla_\theta J^{\text{inner}}} \quad (4)$$

$$\underbrace{\left(\nabla_{\theta'}\log\pi_{\theta'}(\boldsymbol{\tau}')R(\boldsymbol{\tau}')\right)}_{\nabla_{\theta'}J^{\text{outer}}}\Bigg) \quad (5)$$

where, $\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}')$ optimizes θ to increase the likelihood of the trajectories $\boldsymbol{\tau}'$ that lead to higher returns given some trajectories $\boldsymbol{\tau}$. In other words, 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.

ProMP (Rothfuss et al., 2018) analyzes this issue with MAML and incorporates $\nabla_\theta J_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}')$ term in the update as well. They propose to use DICE (Foerster et al., 2018) to allow causal credit assignment on the pre-update trajectory distribution, however, the gradients computed by DICE suffer from high variance estimates. To remedy this, they proposed a low variance (and slightly biased) approximation of the DICE based loss that leads to stable updates.

3. Method

The pre-update and post-update policies are often expected to exhibit very different behaviors, i.e, exploration and exploitation behaviors respectively. 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 using the exploration policy for sampling, we need to perform importance sampling. The update thus becomes:

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

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 ($\nabla_\theta J^{\text{inner}}$,$\nabla^2_\theta \log \pi_\theta(\boldsymbol{\tau})$) terms appearing in Eq. 2 and 5. 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 resulting in stability 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 the appendix.

3.1. 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.

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 $\left(M_{\beta,z}(s,a) - M^t(s,a)\right)^2$:

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

where, $\overline{M}(s,a)$ is the target which can be any of the task specific quantities as mentioned above. We further modify the above equation by multiplying the DICE operator to simplify the gradient computation w.r.t ϕ. This eliminates the need to apply the policy gradient trick to expand the above expression for gradient computation. The 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. After obtaining the updated parameters z', the agent samples trajectories $\boldsymbol{\tau}'$ using its updated exploitation policy $\pi_{\theta,z'}$. Note that our model enables the agent to learn a generic exploitation policy $\pi_{\theta,z}$ for the task distribution which can then be adapted to any specific task by updating z to z' as shown above. Effectively, z' encodes the necessary information regarding the task that helps an agent in adapting its behavior to maximize its expected return. The collected trajectories are then used to perform a policy gradient update for all the parameters z, θ, ϕ and β using the following objective:

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

The gradients of $J(z',\theta)$ w.r.t. ϕ are shown in Eq. 6 (see appendix). Although the gradients are unbiased, they still have very high variance. To solve this problem, we draw inspiration from Mnih et al. (2016) and replace the return R^μ_t (see Eq. 7 in appendix) with an advantage estimate A^μ_t (see 8 in appendix). Due to space constraints we describe these formulations in more detail in appendix.

4. Experiments

We have evaluated our proposed model on the environments used by Rothfuss et al. (2018). Specifically, we have used HalfCheetahFwdBack, HalfCheetahVel and Walker2DFwdBack environments for the dense reward tasks and a 2D point environment proposed in Rothfuss et al. (2018) for sparse rewards.

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 z.

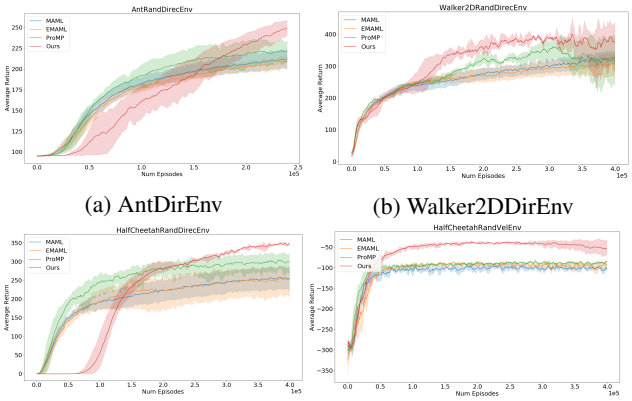
The results of the baselines for the benchmark environments have been borrowed directly from the the official ProMP website [1]. For the point environments, we have used the publicly available official implementation[2].

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

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

4.1. Results

We also compare our method with 3 baseline approaches: MAML, EMAML and ProMP on the benchmark continuous control tasks. The performance plots for all four algorithms are shown in Fig. 1. In all the environments, our proposed method outperforms others in terms of asymptotic performance. The training is also more stable for our method and leads to lower variance plots. Our algorithm particularly shines in 2DPointEnvCorner (Fig 2a) where the reward is sparse. 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. Although ProMP manages to reach similar peak performance to our method in 2DPointEnvCorner, the training itself is pretty unstable indicating the inherent fragility of their updates. Further, we show that our method leads to good exploration behavior in a sparse reward point environment where the agent is allowed to sample only two trajectories in order to perform the updates illustrating the strength of our procedure. We also show that the separation of exploration and exploitation policies in this scenario allows us to train the exploration policy using an independent objective providing better performance in certain situations.



(a) AntDirEnv (b) Walker2DDirEnv

(c) HalfCheetahDirEnv (d) HalfCheetahVelEnv

Figure 1. Comparison of our method with 3 baseline methods in sparse and dense reward environments.

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

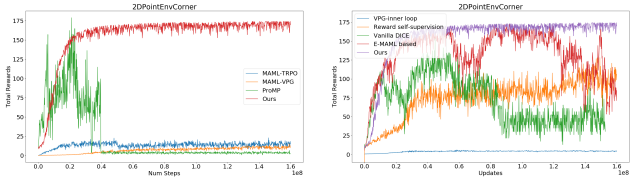
VPG-Inner loop : The supervised loss in the inner loop is replaced with the vanilla policy gradient loss as in MAML while using the exploration policy to sample the pre-update trajectories. As expected, this model performs poorly due to the high variance off-policy updates in the inner loop.

Reward Self-Supervision : A reward based self-supervised objective is used instead of return based self-supervision. This variant performs reasonably well but struggles to reach peak performance since the task is sparse reward.

Vanilla DICE : We directly use the dice gradients to perform updates on ϕ instead of using the low variance gradient estimator. The high variance dice gradients lead to unstable training as can be seen from the plots.

E-MAML Based : Used an E-MAML (Stadie et al., 2018) type objective to compute the gradients w.r.t ϕ instead of using DICE. Although this variant manages to reach peak performance, it is unstable due to the lack of causal credit assignment.

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



(a) Baseline Comparisons (b) Ablation results

Figure 2. 2D Point Environment

5. Conclusion and Future Work

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. Hence, as future work, we would like to explore the use of separate exploration and exploitation policies in other meta-learning approaches as well. We showed that, through various experiments on both sparse and dense reward tasks, our model outperforms previous works while also being very stable during training. This validates that using self-supervised techniques increases the stability of these updates thus allowing us to use a separate exploration policy to collect the initial trajectories. Further, we also show that the variance reduction techniques used in the objective of exploration policy also have a huge impact on the performance. However, we would like to note that the idea of using a separate exploration and exploitation policy is much more general and doesn't need to be restricted to MAML.

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

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

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

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

Gupta, A., Eysenbach, B., Finn, C., and Levine, S. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018a.

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

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

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

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

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

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

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

Stadie, B. C., Yang, G., Houthooft, R., Chen, X., Duan, Y., Wu, Y., Abbeel, P., and Sutskever, I. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

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

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

6. Appendix

We described our proposed model in Section 3. The pseudo-code of our algorithm is shown in algorithm 6.

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 $\left(M_{\beta,z}(s,a) - \bar{M}(s,a)\right)^2$:

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

    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 to optimize all the parameters z, θ, ϕ, β

Although, in principle, the above algorithm is what we want to implement, implementing it as is leads to high variance dice gradients. This consequently leads to high variance gradients for the ϕ's. To alleviate this, we apply some variance reduction techniques described as follows.

The vanilla dice gradients 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 seen 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

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

Thus, instead of directly applying the gradients computed above, we compute the advantages A_t^μ using the returns R_t^μ and apply the following gradients.

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

where,

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

where V_t^μ is computed using using a linear feature baseline (Duan et al., 2016) fitted on the returns R_t^μ. And r_t^μ is given by,

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

6.0.1. 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 (Kulkarni et al., 2016) (similar to N-step returns) $m_\beta(s_t, a_t)$ and a linear layer on top of

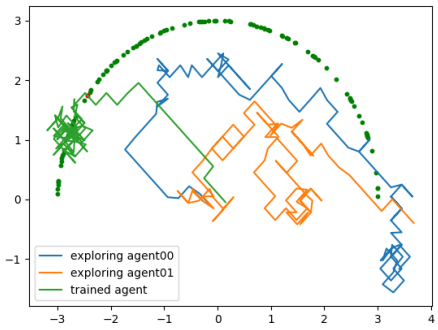


Figure 3. Ours

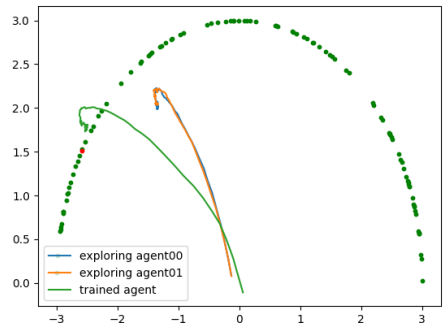


Figure 5. ProMP

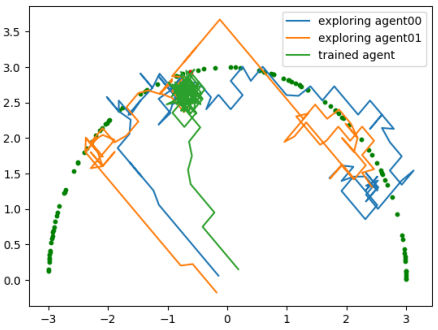


Figure 4. μ_ϕ maximizes its environment rewards

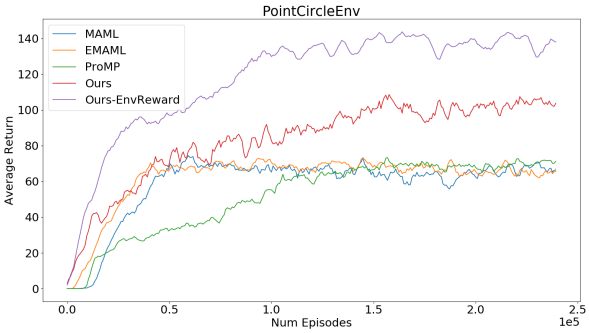


Figure 6. Comparison with baselines

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 $(s_t, a_t)pair$.

6.1. Additional experiments

6.1.1. Point Environment - Semicircle

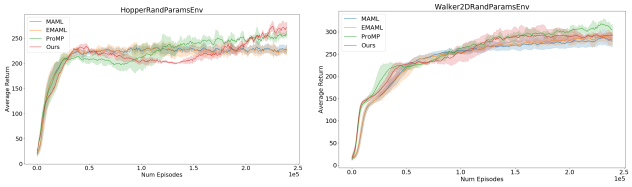
We perform some additional experiments on another toy environment to illustrate the exploration behavior shown by our model and demonstrate the benefits of using different exploration and exploitation policies. Fig 3 shows an environment where the agent is initialized at the center of the semi-circle. 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 3 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. 4 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 5. It is clear that it struggles to learn different exploration and exploitation behaviors. Fig. 6 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.

6.1.2. EXPERIMENTS WITH DYNAMICS BASED VARIATIONS

We also experiment with Walker2DRandParams and HopperRandParams. The different tasks in these environments arise from variations in the dynamics of the agent. The results are shown in Fig 7. We observe that in both these environments we match the performance of the baselines but don't really perform much better. This could be because we still use the n-step return as the self-supervision objective in our experiments. We expect the results to get better if we test with next-state prediction etc as self-superivision objectives. We leave that for future work.

6.2. Hyper-parameters and Details

For all the experiments, we treat the shared parameter z as a learnable latent embedding with fixed initial values of $\bar{0}$ as proposed in (Zintgraf et al., 2018),i.e, we don't perform any outer-loop updates on z.. 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 (Rothfuss et al., 2018) 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 (Kingma & Ba, 2014) optimizer with a learning rate of $7e-4$ for all parameters except ϕ, which uses a learning rate of $7e-5$. 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. We have provided a version of our code in the supplementary material. We will soon open source a cleaned version of this online.



(a) HopperRandParamsEnv (b) Walker2DRandParamsEnv

Figure 7. Comparison of our method with 3 baseline methods in dynamics based environments.