Model-based Adversarial Meta-Reinforcement Learning

Zichuan Lin [1]   **Garrett Thomas** [2]   **Guangwen Yang** [1]   **Tengyu Ma** [2]

Abstract

Meta-reinforcement learning (meta-RL) aims to learn from multiple training tasks the ability to adapt efficiently to unseen test tasks. Despite the success, existing meta-RL algorithms are known to be sensitive to the task distribution shift. When the test task distribution is different from the training task distribution, the performance may degrade significantly. To address this issue, this paper proposes *Model-based Adversarial Meta-Reinforcement Learning* (AdMRL), where we aim to minimize the worst-case sub-optimality gap – the difference between the optimal return and the return that the algorithm achieves after adaptation – across all tasks in a family of tasks, with a model-based approach. We propose a minimax objective and optimize it by alternating between learning the dynamics model on a fixed task and finding the *adversarial* task for the current model – the task for which the policy induced by the model is maximally suboptimal. Assuming the family of tasks is parameterized, we derive a formula for the gradient of the suboptimality with respect to the task parameters via the implicit function theorem, and show how the gradient estimator can be efficiently implemented by the conjugate gradient method and a novel use of the REINFORCE estimator. We evaluate our approach on several continuous control benchmarks and demonstrate its efficacy in the worst-case performance over all tasks, the generalization power to out-of-distribution tasks, and in training and test time sample efficiency, over existing state-of-the-art meta-RL algorithms.

1. Introduction

Deep reinforcement learning (Deep RL) methods can solve difficult tasks such as Go (Silver et al., 2016), Atari games (Mnih et al., 2013), robotic control (Levine et al., 2016) successfully, but often require sampling a large amount interactions with the environment. Meta-reinforcement learning and multi-task reinforcement learning aim to improve the sample efficiency by leveraging the shared structure within a family of tasks. For example, Model Agnostic Meta Learning (MAML) (Finn et al., 2017) learns in the training time a shared policy initialization across tasks, from which in the test time it can adapt to the new tasks quickly with a small amount of samples. The more recent work PEARL (Rakelly et al., 2019) learns latent representations of the tasks in the training time, and infers the representations of test tasks and adapts to them.

The existing meta-RL formulation and methods are largely *distributional*. The training tasks and the testing tasks are assumed to be drawn from the same distribution of tasks. Consequently, the existing methods are prone to the distribution shift issue, as shown in (Mehta et al., 2020) — when the tasks in the test time are not drawn from the same distribution as in the training, the performance degrades significantly. Figure 1 also confirms this issue for PEARL (Rakelly et al., 2019), a recent state-of-the-art meta-RL method, on the `Ant2D-velocity` tasks. PEARL can adapt to tasks with smaller goal velocities much better than tasks with larger goal velocities, in terms of the relative difference, or the sub-optimality gap, from the optimal policy of the corresponding task.[1] To address this issue, Mehta et al. (2020) propose an algorithm that iteratively re-define the task distribution to focus more on the hard task.

In this paper, we instead take a *non-distributional* perspective by formulating the *adversarial* meta-RL problem. Given a parametrized family of tasks, we aim to minimize the worst sub-optimality gap — the difference between the optimal return and the return the algorithm achieves after adaptation — across all tasks in the family in the test time. This can be naturally formulated mathematically as a minimax problem (or a two-player game) where the maximum is over all the tasks and the minimum is over the parameters of the algorithm (e.g., the shared policy initialization or the shared dynamics).

[1]Tsinghua University [2]Stanford University. Correspondence to: Zichuan Lin <linzc16@mails.tsinghua.edu.cn>.

---

[1]The same conclusion is still true if we measure the raw performance on the tasks. But that could be misleading because the tasks have varying optimal returns.
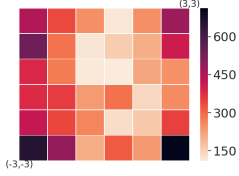
Figure 1. The performance of PEARL (Rakelly et al., 2019) on `Ant2D-velocity` tasks. Each task is represented by the target velocity $(x, y) \in \mathbb{R}^2$ with which the ant should run. The training tasks are uniformly drawn in $[-3, 3]^2$. The color of each cell shows the sub-optimality gap of the corresponding task, namely, the optimal return of that task minus the return of PEARL. Lighter means smaller sub-optimality gap and is better. High-velocity tasks tend to perform worse, which implies that if the test task distribution shift towards high-velocity tasks, the performance will degrade.

Our approach is model-based. We learn a shared dynamics model across the tasks in the training time, and during the test time, given a new reward function, we train a policy on the learned dynamics. The model-based methods can outperform significantly the model-free methods in sample-efficiency even in the standard single task setting (Luo et al., 2018; Dong et al., 2019; Janner et al., 2019; Wang & Ba, 2019; Chua et al., 2018; Buckman et al., 2018; Nagabandi et al., 2018c; Kurutach et al., 2018; Feinberg et al., 2018; Rajeswaran et al., 2016; 2020; Wang et al., 2019), and are particularly suitable for meta-RL settings where the optimal policies for tasks are very different, but the underlying dynamics is shared (Landolfi et al., 2019). We apply the natural adversarial training (Madry et al., 2017) on the level of tasks — we alternate between the minimizing the sub-optimality gap over the parameterized dynamics and maximizing it over the parameterized tasks.

The main technical challenge is to optimize over the task parameters in a sample-efficient way. The sub-optimality gap objective depends on the task parameters in a non-trivial way because the algorithm uses the task parameters iteratively in its adaptation phase during the test time. The naive attempt to back-propagate through the sequential updates of the adaptation algorithm is time costly, especially because the adaptation time in the model-based approach is computationally expensive (despite being sample-efficient). Inspired by a recent work on learning equilibrium models in supervised learning (Bai et al., 2019), we derive an efficient formula of the gradient w.r.t. the task parameters via the implicit function theorem. The gradient involves an inverse Hessian vector product, which can be efficiently computed by conjugate gradients and the REINFORCE estimator (Williams, 1992). In summary, our contributions are: (1) We propose a minimax formulation of model-based adversarial meta-reinforcement learning (AdMRL, pronounced like "admiral") with an adversarial training algorithm to address the distribution shift problem. (2) We derive an estimator of the gradient with respect to the task parameters, and show how it can be implemented efficiently in both samples and time. (3) Our approach significantly outperforms the state-of-the-art meta-RL algorithms in the worst-case performance over all tasks, the generalization power to out-of-distribution tasks, and in training and test time sample efficiency on a set of continuous control benchmarks.

2. Preliminaries

Reinforcement Learning. Consider a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, T, r, p_0, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are state

and action space, T is transition dynamics, r is reward function, $\gamma \in [0, 1)$ is discount factor and p_0 is an initial state distribution. Let $\pi(\cdot|s)$ be policy. Let $T^\star$ denote the unknown true transition dynamics. We define the value function $V^{\pi,T} : \mathcal{S} \to \mathbb{R}$ at state s for a policy π on dynamics T: $V^{\pi,T}(s) = \mathop{\mathbb{E}}_{a_t, s_t \sim \pi, T} [\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s]$. The goal of RL is to seek a policy that maximizes the expected return $\eta(\pi, T) := \mathop{\mathbb{E}}_{s_0 \sim p_0} [V^{\pi,T}(s_0)]$.

Meta-Reinforcement Learning. In this paper, we consider a family of tasks parameterized by $\Psi \subseteq \mathbb{R}^k$ and a family of polices parameterized by $\Theta \subseteq \mathbb{R}^p$. The family of tasks is a family of MDP $\{(\mathcal{S}, \mathcal{A}, T, r_\psi, p_0, \gamma)\}_{\psi \in \Psi}$ which all share the same dynamics but differ in the reward function. We denote the value function of a policy π on a task with reward r_ψ and dynamics T by $V_\psi^{\pi,T}$, and denote the expected return for each task and dynamics by $\eta(\pi, T, \psi) = \mathbb{E}[V_\psi^{\pi,T}(s_0)]$. For simplicity, we will use the shorthand $\eta(\theta, T, \psi) := \eta(\pi_\theta, T, \psi)$. Let $\Phi \subseteq \mathbb{R}^d$ denote the set of all structures. A meta-RL training algorithm seeks to find a shared structure $\phi \in \Phi$, which is subsequently used by an adaptation algorithm $A : \Phi \times \Psi \to \Theta$ to learn quickly in new tasks. In this paper, the shared structure ϕ is the learned dynamics (more below).

Model-based Reinforcement Learning. In model-based reinforcement learning (MBRL), we parameterize the transition dynamics of the model $\widehat{T}_\phi$ and learn the parameters ϕ so that it approximates the true transition dynamics of $T^\star$. In this paper, we use Stochastic Lower Bound Optimization (SLBO) (Luo et al., 2018), which is an MBRL algorithm with theoretical guarantees of monotonic improvement.

3. Model-based Adversarial Meta-Reinforcement Learning

Formulation. We consider a family of tasks whose reward functions $r_\psi(s, a)$ are parameterized by some parameters ψ, and assume that $r_\psi(s, a)$ is differentiable w.r.t. ψ for every s, a. We assume the reward function parameterization $r_\psi(\cdot, \cdot)$ is known throughout the paper.[2] Recall that the total return of policy π_θ on dynamics T and tasks ψ is denoted by $\eta(\theta, T, \psi) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta, T} [R_\psi(\tau)]$. Here $R_\psi(\tau)$ is the return of the trajectory under reward function r_ψ. As shorthand, we define $\eta^\star(\theta, \psi) = \eta(\theta, T^\star, \psi)$ as the return in the real

---

[2]It's challenging to formulate the worst-case performance without knowing a reward family, e.g., when we only have access to randomly sampled tasks from a task distribution.

environment on tasks ψ and $\hat{\eta}_\phi(\theta, \psi) = \eta(\theta, \widehat{T}_\phi, \psi)$ as the return on the virtual dynamics $\widehat{T}_\phi$ on task ψ.

Given a learned dynamics $\widehat{T}_\phi$ and test task ψ, we can perform a *zero-shot model-based adaptation* by computing the best policy for task ψ under the dynamics $\widehat{T}_\phi$, namely, $\arg\max_\theta \hat{\eta}_\phi(\theta, \psi)$. Let $\mathcal{L}(\phi, \psi)$, formally defined in equation below, be the *suboptimality gap* of the $\widehat{T}_\phi$-optimal policy on task ψ, i.e. the difference between the performance of the best policy for task ψ and the performance of the policy which is best for ψ according to the model $\widehat{T}_\phi$. Our overall aim is to find the best shared dynamics $\widehat{T}_\phi$, such that the worst-case sub-optimality gap $\mathcal{L}(\phi, \psi)$ is minimized. This can be formally written as a minimax problem:

$$\min_\phi \max_\psi \underbrace{\left[\max_\theta \eta^\star(\theta, \psi) - \eta^\star\left(\arg\max_\theta \hat{\eta}_\phi(\theta, \psi), \psi\right) \right]}_{\triangleq \mathcal{L}(\phi, \psi)}. \quad (1)$$

In the inner step (max over ψ), we search for the task ψ which is hardest for our current model $\widehat{T}_\phi$, in the sense that the policy which is optimal under dynamics $\widehat{T}_\phi$ is most suboptimal in the real MDP. In the outer step (min over $\widehat{T}_\phi$), we optimize for a model with low worst-case suboptimality. We remark that, in general, other definitions of sub-optimality gap, e.g., the ratio between the optimal return and achieved return may also be used to formulate the problem.

Algorithmically, by training on the hardest task found in the inner step, we hope to obtain data that is most informative for correcting the model's inaccuracies.

Computing Derivatives with respect to Task Parameters. To optimize Eq. (1), we will alternate between the min and max using gradient descent and ascent respectively. Fixing the task ψ, minimizing $\mathcal{L}(\phi, \psi)$ reduces to standard MBRL.

On the other hand, for a fixed model $\widehat{T}_\phi$, the inner maximization over the task parameter ψ is non-trivial, and is the focus of this subsection. To perform gradient-based optimization, we need to estimate $\frac{\partial \mathcal{L}}{\partial \psi}$. Let us define $\theta^\star = \arg\max_\theta \eta^\star(\theta, \psi)$ (the optimal policy under the true dynamics and task ψ) and $\hat{\theta} = \arg\max_\theta \hat{\eta}_\phi(\theta, \psi)$ (the optimal policy under the virtual dynamics and task ψ). We assume there is a unique $\hat{\theta}$ for each ψ. Then,

$$\frac{\partial \mathcal{L}}{\partial \psi} = \frac{\partial \eta^\star}{\partial \psi}\Big|_{\theta^\star} - \left(\frac{\partial \hat{\theta}^\top}{\partial \psi} \frac{\partial \eta^\star}{\partial \theta}\Big|_{\hat{\theta}} + \frac{\partial \eta^\star}{\partial \psi}\Big|_{\hat{\theta}} \right). \quad (2)$$

Note that the first term comes from the usual (sub)gradient rule for pointwise maxima, and the second term comes from the chain rule. Differentiation w.r.t. ψ commutes with expectation over τ, so

$$\frac{\partial \eta^\star}{\partial \psi} = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta, T^\star} \left[\frac{\partial R_\psi(\tau)}{\partial \psi} \right] = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta, T^\star} \left[\sum_{t=0}^\infty \gamma^t \frac{\partial r_\psi(s_t, a_t)}{\partial \psi} \right]. \quad (3)$$

Thus the first and last terms of the gradient of Eq. (2) can be estimated by simply rolling out $\pi_{\theta^\star}$ and $\pi_{\hat{\theta}}$ and

differentiating the sampled rewards. Let $A^{\pi_{\hat{\theta}}}(s_t, a_t)$ be the advantage function. Then, the term $\frac{\partial \eta^\star}{\partial \theta}\Big|_{\hat{\theta}}$ in Eq. (2) can be computed by the standard policy gradient

$$\frac{\partial \eta^\star}{\partial \theta}\Big|_{\hat{\theta}} = \mathop{\mathbb{E}}_{\tau \sim \pi_{\hat{\theta}}, T^\star} \left[\sum_{t=0}^\infty \gamma^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta}\Big|_{\hat{\theta}} A^{\pi_{\hat{\theta}}}(s_t, a_t) \right]. \quad (4)$$

The complicated part left in Eq. (2) is $\frac{\partial \hat{\theta}^\top}{\partial \psi}$. We compute it using the implicit function theorem (Wikipedia contributors, 2020) (see Section A.1 for details):

$$\frac{\partial \hat{\theta}}{\partial \psi^\top} = -\left(\frac{\partial^2 \hat{\eta}_\phi}{\partial \theta \partial \theta^\top}\Big|_{\hat{\theta}} \right)^{-1} \frac{\partial^2 \hat{\eta}_\phi}{\partial \theta \partial \psi^\top}\Big|_{\hat{\theta}}. \quad (5)$$

The mixed-derivative term in equation above can be computed by differentiating the policy gradient:

$$\frac{\partial^2 \hat{\eta}_\phi}{\partial \theta \partial \psi^\top}\Big|_{\hat{\theta}} = \mathop{\mathbb{E}}_{\tau \sim \pi_{\hat{\theta}}, \widehat{T}_\phi} \left[\sum_{t=0}^\infty \gamma^t \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta}\Big|_{\hat{\theta}} \frac{\partial A^{\pi_{\hat{\theta}}}(s_t, a_t)}{\partial \psi^\top} \right]. \quad (6)$$

An estimator for the Hessian term in Eq. (5) can be derived by the REINFORCE estimator (Sutton et al., 2000), or the log derivative trick (see Section A.2 for a detailed derivation),

$$\frac{\partial^2 \hat{\eta}_\phi}{\partial \theta \partial \theta^\top} = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta, \widehat{T}_\phi} \left[\left(\frac{\partial \log \pi_\theta(\tau)}{\partial \theta} \frac{\partial \log \pi_\theta(\tau)}{\partial \theta^\top} + \frac{\partial^2 \log \pi_\theta(\tau)}{\partial \theta \partial \theta^\top} \right) R_\psi(\tau) \right]. \quad (7)$$

By computing the gradient estimator using implicit function theorem, we do not need to back-propagate through the sequential updates of our adaptation algorithm, from which we can estimate the gradient w.r.t. task parameters in a sample-efficient and computationally tractable way. We provide a practical implementation of our algorithm AdMRL in Appendix B.

4. Experiments

In our experiments, we aim to study the following questions: (1) How does AdMRL perform on standard meta-RL benchmarks compared to prior state-of-the-art approaches? (2) Does AdMRL achieve better worst-case performance than *distributional* meta-RL methods? (3) How does AdMRL perform in environments where task parameters are high-dimensional? (4) Does AdMRL generalize better than distributional meta-RL on out-of-distribution tasks?

4.1. Adaptation Performance Compared to Baselines

We evaluate our approach on a variety of continuous control tasks based on OpenAI gym (Brockman et al., 2016), which uses the MuJoCo physics simulator (Todorov et al., 2012). We list all experiments setup in Appendix C.

We compare our algorithm against MAML and PEARL. Figure 2 shows the adaptation results on the testing tasks set. We produce the curves by: (1) running our algorithm and baseline algorithms by training on adversarially chosen tasks and uniformly sampling random tasks respectively; (2)
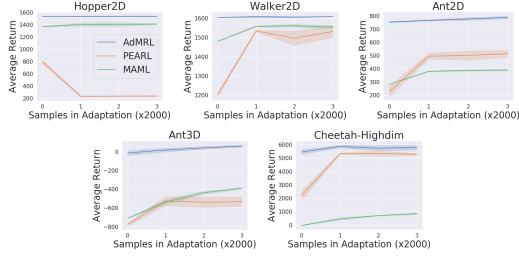
Figure 2. Average of returns $A_n(\psi)$ over all tasks of adapted policies (with 3 random seeds) from AdMRL, MAML and PEARL. Our approach substantially outperforms baselines in training and test time sample efficiency, and even with zero-shot adaptation.
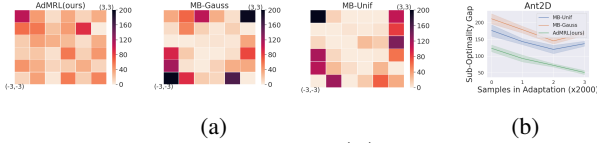

(a)   (b)

Figure 3. (a) Sub-optimality gap $G_n(\psi)$ of adapted policies $n = 6K$ for each test task ψ from AdMRL, MB-Unif, and MB-Gauss. Lighter means smaller, which is better. For tasks on the boundary, AdMRL achieves much lower $G_n(\psi)$ than MB-Gauss and MB-Unif, which indicates AdMRL generalizes better in the worst case. (b) The worst-case sub-optimality gap $G_n^{\max}$ in the number of adaptation samples n. AdMRL successfully minimizes the worst-case suboptimality gap.

for each test task, we first do zero-shot adaptation for our algorithm and then run our algorithm and baseline algorithms by collecting samples; (3) estimating the averaged returns of the policies by sampling new roll-outs. The curves show the return averaged across all testing tasks with three random seeds in testing time. Our approach AdMRL outperforms MAML and PEARL across all test tasks, even though our method visits much fewer tasks (7/8) and samples (2/3) than baselines during meta-training. AdMRL outperforms MAML and PEARL with even zero-shot adaptation, namely, collecting no samples.[3]

4.2. Comparing with Model-based Baselines in Worst-case Sub-optimality Gap

To investigate the worst-case performance of our approach, we compare our adversarial selection method with *distributional* variants — using model-based training but sampling tasks with a uniform or gaussian distribution with variance 1, denoted by **MB-Unif** and **MB-Gauss**, respectively. We plot heatmap figures by computing the sub-optimality gap for each test task in figure 3(a). We find that while both MB-Gauss and MB-Unif tend to over-fit on the tasks in the center, AdMRL can generalize much better to the tasks on the boundary. In figure 3(b), We also find that AdMRL can achieve lower sub-optimality gap in the worst cases.

---

[3]Note that the zero-shot model-based adaptation is taking advantage of additional information (the reward function) which MAML and PEARL have no mechanism for using.
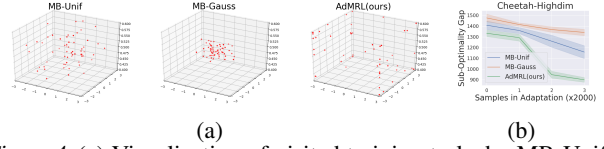

(a)   (b)

Figure 4. (a) Visualization of visited training tasks by MB-Unif, MB-Gauss and AdMRL; AdMRL can quickly visit tasks with large suboptimality gap on the boundary and train the model to minimize the worst-case suboptimality gap. (b) The worst-case suboptimality gap $G_n^{\max}$ in the number of adaptation samples n for high-dimensional tasks. AdMRL significantly outperforms baselines in such tasks.

Performance on high-dimensional tasks. We show in figure 4(b) that AdMRL performs significantly better than MB-Unif and MB-Gauss when the task parameters are high-dimensional. In the high-dimensional tasks, sampling from a given distribution of tasks during meta-training becomes less efficient — it is hard to cover all tasks with worst suboptimality gap by randomly sampling from a given distribution. On the contrary, our non-distributional adversarial selection way can search for those hardest tasks efficiently and train a model that minimizes the worst suboptimality gap.

Visualization. To understand how our algorithm works, we visualize the task parameter ψ that visited during meta-training in `Ant3D` environment. We compare our method with MB-Unif and MB-Gauss in figure 4(a). We find that our method can quickly visit the *hard* tasks on the boundary, in the sense that we can find the most informative tasks to train our model. On the contrary, sampling randomly from uniform or gaussian distribution has much less probability to visit the tasks on the boundary.

4.3. Out-of-distribution Performance

We evaluate our algorithm on out-of-distribution tasks in the `Ant2D` environment. We train agents with tasks drawn in $\Psi = [-3, 3]^2$ while testing on OOD tasks from $\Psi = [-5, 5]^2$. Appendix E shows that AdMRL has much lower suboptimality gap than MB-Unif and MB-Gauss on OOD tasks, which shows the generalization power of AdMRL.

5. Conclusion

In this paper, we propose Model-based Adversarial Meta-Reinforcement Learning (AdMRL), to address the distribution shift issue of meta-RL. We formulate the *adversarial meta-RL problem* and propose a minimax formulation to minimize the worst sub-optimality gap. To optimize efficiently, we derive an estimator of the gradient with respect to the task parameters, and implement the estimator efficiently using the conjugate gradient method. We provide extensive results on standard benchmark environments to show the efficacy of our approach over prior meta-RL algorithms. In the future, several interesting directions lie ahead. (1) Apply AdMRL to more difficult settings such as visual domain. (2) Use other MBRL algorithms. (3) Apply AdMRL to cases where the parameterization of reward function is unknown.

References

Atlas, L. E., Cohn, D. A., and Ladner, R. E. Training connectionist networks with queries and selective sampling. In *Advances in neural information processing systems*, pp. 566–573, 1990.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pp. 688–699, 2019.

Bengio, S., Bengio, Y., Cloutier, J., and Gecsei, J. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, volume 2. Univ. of Texas, 1992.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pp. 8224–8234, 2018.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018a.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018b.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4754–4765, 2018.

Dong, K., Luo, Y., and Ma, T. Bootstrapping the expressivity with model-based planning. *arXiv preprint arXiv:1910.05927*, 2019.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Fakoor, R., Chaudhari, P., Soatto, S., and Smola, A. J. Meta-q-learning. *arXiv preprint arXiv:1910.00125*, 2019.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

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.

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

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

Humplik, J., Galashov, A., Hasenclever, L., Ortega, P. A., Teh, Y. W., and Heess, N. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pp. 12498–12509, 2019.

Jin, C., Krishnamurthy, A., Simchowitz, M., and Yu, T. Reward-free exploration for reinforcement learning. *arXiv preprint arXiv:2002.02794*, 2020.

Kirsch, L., van Steenkiste, S., and Schmidhuber, J. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Lan, L., Li, Z., Guan, X., and Wang, P. Meta reinforcement learning with task embedding and shared policy. *arXiv preprint arXiv:1905.06527*, 2019.

Landolfi, N. C., Thomas, G., and Ma, T. A model-based approach for sample-efficient multi-task reinforcement learning. *arXiv preprint arXiv:1907.04964*, 2019.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Lewis, D. D. and Gale, W. A. A sequential algorithm for training text classifiers. In *SIGIR'94*, pp. 3–12. Springer, 1994.

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Mehta, B., Deleu, T., Raparthy, S. C., Pal, C. J., and Paull, L. Curriculum in gradient-based meta-reinforcement learning. *arXiv preprint arXiv:2002.07956*, 2020.

Mendonca, R., Gupta, A., Kralev, R., Abbeel, P., Levine, S., and Finn, C. Guided meta-policy search. In *Advances in Neural Information Processing Systems*, pp. 9653–9664, 2019.

Mendonca, R., Geng, X., Finn, C., and Levine, S. Meta-reinforcement learning robust to distributional shift via model identification and experience relabeling. *arXiv preprint arXiv:2006.07178*, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018a.

Nagabandi, A., Finn, C., and Levine, S. Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*, 2018b.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018c.

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

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

Rajeswaran, A., Mordatch, I., and Kumar, V. A game theoretic framework for model based reinforcement learning. *arXiv preprint arXiv:2004.07804*, 2020.

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

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

Sæmundsson, S., Hofmann, K., and Deisenroth, M. P. Meta reinforcement learning with latent variable gaussian processes. *arXiv preprint arXiv:1803.07551*, 2018.

Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook.* PhD thesis, Technische Universität München, 1987.

Schulze, S., Whiteson, S., Zintgraf, L., Igl, M., Gal, Y., Shiarlis, K., and Hofmann, K. Varibad: a very good method for bayes-adaptive deep rl via meta-learning. International Conference on Learning Representations.

Settles, B. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

Silberman, M. *Active Learning: 101 Strategies To Teach Any Subject.* ERIC, 1996.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pp. 4077–4087, 2017.

Sutton, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pp. 216–224. Elsevier, 1990.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Thrun, S. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pp. 640–646, 1996.

Thrun, S. and Pratt, L. *Learning to learn.* Springer Science & Business Media, 2012.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Utgoff, P. E. Shift of bias for inductive concept learning. *Machine learning: An artificial intelligence approach*, 2: 107–148, 1986.

Wang, H., Zhou, J., and He, X. Learning context-aware task reasoning for efficient meta-reinforcement learning. *arXiv preprint arXiv:2003.01373*, 2020.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Wang, T. and Ba, J. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P., and Ba, J. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Wikipedia contributors. Implicit function theorem — Wikipedia, the free encyclopedia, 2020. URL `https://en.wikipedia.org/w/index.php?title=Implicit_function_theorem&oldid=953711659`. [Online; accessed 2-June-2020].

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Zintgraf, L., Igl, M., Shiarlis, K., Mahajan, A., Hofmann, K., and Whiteson, S. Variational task embeddings for fast adapta-tion in deep reinforcement learning. In *International Conference on Learning Representations Workshop on Structure & Priors in Reinforcement Learning*, 2019.

A. Omitted Derivations

A.1. Jacobian of $\hat{\theta}$ with respect to ψ

We begin with an observation: first-order optimality conditions for $\hat{\theta}$ necessitate that

$$\left.\frac{\partial \hat{\eta}}{\partial \theta}\right|_{\hat{\theta}} = 0 \tag{8}$$

Then, the implicit function theorem tells us that for sufficiently small $\Delta\psi$, there exists $\Delta\theta$ as a function of $\Delta\psi$ such that

$$\left.\frac{\partial \hat{\eta}}{\partial \theta}\right|_{\hat{\theta}+\Delta\theta, \psi+\Delta\psi} = 0 \tag{9}$$

To first order, we have

$$\left.\frac{\partial \hat{\eta}}{\partial \theta}\right|_{\hat{\theta}+\Delta\theta, \psi+\Delta\psi} \approx \underbrace{\left.\frac{\partial \hat{\eta}}{\partial \theta}\right|_{\hat{\theta}}}_{0} + \left.\frac{\partial^2 \hat{\eta}}{\partial\theta\partial\theta^\top}\right|_{\hat{\theta}} \Delta\theta + \left.\frac{\partial^2 \hat{\eta}}{\partial\theta\partial\psi^\top}\right|_{\hat{\theta}} \Delta\psi \tag{10}$$

Thus, solving for $\Delta\theta$ as a function of $\Delta\psi$ and taking the limit as $\Delta\psi \to 0$, we obtain

$$\frac{\partial \hat{\theta}}{\partial \psi^\top} = -\left(\left.\frac{\partial^2 \hat{\eta}}{\partial\theta\partial\theta^\top}\right|_{\hat{\theta}}\right)^{-1} \left.\frac{\partial^2 \hat{\eta}}{\partial\theta\partial\psi^\top}\right|_{\hat{\theta}} \tag{11}$$

A.2. Policy Hessian

Fix dynamics T, and let $\pi_\theta(\tau)$ denote the probability density of trajectory τ under policy π_θ. Then we have

$$\frac{\partial \log \pi_\theta(\tau)}{\partial \theta} = \frac{\frac{\partial \pi_\theta(\tau)}{\partial \theta}}{\pi_\theta(\tau)} \tag{12}$$

$$\text{i.e.} \quad \frac{\partial \pi_\theta(\tau)}{\partial \theta} = \pi_\theta(\tau) \frac{\partial \log \pi_\theta(\tau)}{\partial \theta} \tag{13}$$

Thus we get the basic (REINFORCE) policy gradient

$$\frac{\partial \eta}{\partial \theta} = \frac{\partial}{\partial \theta} \int \pi_\theta(\tau) R_\psi(\tau)\,\mathrm{d}\tau \tag{14}$$

$$= \int \frac{\partial \pi_\theta(\tau)}{\partial \theta} R_\psi(\tau)\,\mathrm{d}\tau \tag{15}$$

$$= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta, T}\left[\frac{\partial \log \pi_\theta(\tau)}{\partial \theta} R_\psi(\tau)\right]. \tag{16}$$

Differentiating our earlier expression for $\frac{\partial \pi_\theta(\tau)}{\partial \theta}$ once more, and then reusing that same expression again, we have

$$\frac{\partial^2 \pi_\theta(\tau)}{\partial\theta\partial\theta^\top} \tag{17}$$

$$= \frac{\partial \pi_\theta(\tau)}{\partial \theta}\frac{\partial \log \pi_\theta(\tau)}{\partial \theta^\top} + \pi_\theta(\tau)\frac{\partial^2 \log \pi_\theta(\tau)}{\partial\theta\partial\theta^\top} \tag{18}$$

$$= \pi_\theta(\tau)\left(\frac{\partial \log \pi_\theta(\tau)}{\partial \theta}\frac{\partial \log \pi_\theta(\tau)}{\partial \theta^\top} + \frac{\partial^2 \log \pi_\theta(\tau)}{\partial\theta\partial\theta^\top}\right) \tag{19}$$

Thus

$$\frac{\partial^2 \eta}{\partial\theta\partial\theta^\top} \tag{20}$$

$$= \frac{\partial^2}{\partial\theta\partial\theta^\top} \int \pi_\theta(\tau) R_\psi(\tau)\,\mathrm{d}\tau \tag{21}$$

$$= \int \frac{\partial^2 \pi_\theta(\tau)}{\partial\theta\partial\theta^\top} R_\psi(\tau)\,\mathrm{d}\tau \tag{22}$$

$$= \int \pi_\theta(\tau)\left(\frac{\partial \log \pi_\theta(\tau)}{\partial \theta}\frac{\partial \log \pi_\theta(\tau)}{\partial \theta^\top} + \frac{\partial^2 \log \pi_\theta(\tau)}{\partial\theta\partial\theta^\top}\right) R_\psi(\tau)\,\mathrm{d}\tau \tag{23}$$

$$= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta, T}\left[\left(\frac{\partial \log \pi_\theta(\tau)}{\partial \theta}\frac{\partial \log \pi_\theta(\tau)}{\partial \theta^\top} + \frac{\partial^2 \log \pi_\theta(\tau)}{\partial\theta\partial\theta^\top}\right) R_\psi(\tau)\right]. \tag{24}$$

B. AdMRL: a Practical Implementation

Algorithm 1 gives pseudo-code for our algorithm AdMRL, which alternates the updates of dynamics $\widehat{T}_\phi$ and tasks ψ. Let VirtualTraining($\theta, \phi, \psi, \mathcal{D}, n$) be the shorthand for the procedure of learning a dynamics ϕ using data $\mathcal{D}$ and then optimizing a policy from initialization θ on tasks ψ under dynamics ϕ with n virtual steps. Here parameterized arguments of the procedure are referred to by their parameters (so that the resulting policy, dynamics, are written in θ and ϕ). For each training task parameterized by ψ, we first initialize the policy randomly, and optimize a policy on the learned dynamics until convergence (Line 5), which we refer to as *zero-shot adaptation*. We then use the obtained policy $\pi_{\hat{\theta}}$ to collect data from real environment and perform the MBRL algorithm SLBO (Luo et al., 2018) by interleaving collecting samples, updating models and optimizing policies (Line 6). After collecting samples and performing SLBO updates, we then get an nearly optimal policy $\pi_{\theta^\star}$.

Then we update the task parameter by gradient ascent. With the policy $\pi_{\hat{\theta}}$ and $\pi_{\theta^\star}$, we compute each gradient component (Line 10, 11) and obtain the gradient w.r.t task parameters (Line 12) and perform gradient ascent for the task parameter ψ (Line 13). Now we complete an outer-iteration. Note that for the first training task, we skip the zero-shot adaptation phase and only perform SLBO updates because our dynamical model is untrained. Moreover, because the zero-shot adaptation step is not done, we cannot technically perform our tasks update either because the tasks derivative depends on $\pi_{\hat{\theta}}$, the result of zero-shot adaption (Line 9).

Implementation Details. Computing Eq. (5) for each dimension of ψ involves an inverse-Hessian-vector product. We note that we can compute Eq. (5) by approximately solving the equation $Ax = b$, where A is $\left.\frac{\partial^2 \hat{\eta}}{\partial\theta\partial\theta^\top}\right|_{\hat{\theta}}$ and b is $\left.\frac{\partial^2 \hat{\eta}}{\partial\theta\partial\psi^\top}\right|_{\hat{\theta}}$. However, in large-scale problems (e.g. θ has thousands of dimensions), it is costly (in computation and memory) to form the full matrix A. Instead, the conjugate gradient method provides a way to approximately solve the equation $Ax = b$ without forming the full matrix of A, provided we can compute the mapping $x \mapsto Ax$. The corresponding Hessian-vector product can be computed as efficiently as evaluating the loss function (Pearlmutter, 1994) up to a universal multiplicative factor. Please refer to Appendix B to see how to implement it concretely. In practice, we found that the matrix of A is always not positive-definite, which hinders the convergence of conjugate gradient method. Therefore, we turn to solve the equivalent equation $A^\top Ax = A^\top b$.

The section discusses how to compute Ax using standard automatic differentiation packages. We first define the fol-

---

Algorithm 1 AdMRL: Model-based Adversarial Meta-Reinforcement Learning

1: Initialize model parameter ϕ, task parameter ψ and dataset $\mathcal{D} \leftarrow \emptyset$
2: **for** n_{tasks} iterations **do**
3:     Initialize policy parameter θ randomly
4:     **if** $\mathcal{D} \neq \emptyset$ **then**
5:         $\hat{\theta} = $ VirtualTraining($\theta, \phi, \psi, \mathcal{D}, n_{zeroshot}$)
6:     **for** n_{slbo} iterations **do**
7:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{\, n_{collect}$ collected samples on the real environments $T^\star$ using π_θ with noise $\}$
8:         $\theta^\star = $ VirtualTraining($\theta, \phi, \psi, \mathcal{D}, n_{inner}$)
9:     **if** first task **then** randomly re-initialize ψ; **otherwise then**
10:         Compute gradients $\frac{\partial \eta^\star}{\partial \psi}|_{\theta^\star}$ and $\frac{\partial \eta^\star}{\partial \psi}|_{\hat{\theta}}$ using Eq. 3; compute $\frac{\partial \eta^\star}{\partial \theta}|_{\hat{\theta}}$ using Eq. 4; compute $\frac{\partial^2 \hat{\eta}}{\partial\theta\partial\psi^\top}|_{\hat{\theta}}$ using Eq. 6; compute $\frac{\partial^2 \hat{\eta}_\phi}{\partial\theta\partial\theta^\top}$ using Eq. 7.
11:         Efficiently compute $\frac{\partial \hat{\theta}}{\partial \psi^\top}$ using conjugate gradient method. (see Section B)
12:         Compute the final gradient $\frac{\partial \mathcal{L}}{\partial \psi} = \frac{\partial \eta^\star}{\partial \psi}|_{\theta^\star} - (\frac{\partial \hat{\theta}^\top}{\partial \psi}\frac{\partial \eta^\star}{\partial \theta}|_{\hat{\theta}} + \frac{\partial \eta^\star}{\partial \psi}|_{\hat{\theta}})$
13:         Perform task parameters projected gradient ascent $\psi \leftarrow \Pi_\Psi(\psi + \alpha \frac{\partial \mathcal{L}}{\partial \psi})$

---

lowing function:

$$\eta_h(\theta_1, \theta_2, \theta_3, \theta, \widehat{T}_\phi, \psi) \tag{25}$$
$$= \mathop{\mathbb{E}}_{\pi_\theta, \widehat{T}_\phi} [(\log \pi_{\theta_1}(a_t|s_t) \log \pi_{\theta_2}(a_t|s_t) + \log \pi_{\theta_3}(a_t|s_t))R_\psi(\tau)], \tag{26}$$

where $\theta_1, \theta_2, \theta_3$ are parameter copies of θ. We then use Hessian-vector product to avoid directly computing the second derivatives. Specifically, we compute the two parts in Eq. (7) respectively by first differentiating η_h w.r.t θ_1 and $\theta_2^\top$

$$g_1 = \frac{\partial}{\partial\theta_1}\left(\frac{\partial\eta_h}{\partial\theta_2^\top} \cdot x\right) = \frac{\partial\log\pi_{\theta_1}(a_t|s_t)}{\partial\theta_1}\left(\frac{\partial\log\pi_{\theta_2}(a_t|s_t)}{\partial\theta_2^\top} \cdot x\right)R_\psi(\tau), \tag{27}$$

and then differentiate η_h w.r.t θ_3 for twice

$$g_2 = \frac{\partial}{\partial\theta_3}\left(\frac{\partial\eta_h}{\partial\theta_3^\top} \cdot x\right) = \frac{\partial}{\partial\theta_3}\left(\frac{\partial\log\pi_{\theta_3}(a_t|s_t)}{\partial\theta_3^\top} \cdot x\right)R_\psi(\tau), \tag{28}$$

and thus we have $Ax = g_1 + g_2$.

In terms of time complexity, computing the gradient w.r.t task parameters is quite efficient compared to other steps. On one hand, in each task iteration, for the MBRL algorithm, we need to collect samples for dynamical model fitting, and then rollout m virtual samples using the learned dynamical model for policy update to solve the task, which takes $O(m(d_\phi + d_\theta))$ time complexity, where d_ϕ and d_θ denote the dimensionality of ϕ and θ. On the other hand, we only

need to update the task parameter once in each task iteration, which takes $O(d_\psi d_\theta)$ time complexity by using conjugate gradient descent, where d_ψ denotes the dimensionality of ψ. In practice, for MBRL algorithm, we often need a large amount of virtual samples m (e.g., millions of) to solve the tasks. In the meantime, the dimension of task parameter d_ψ is a small constant and we have $d_\theta \ll d_\phi$. Therefore, in our algorithm, the runtime of computing gradient w.r.t task parameters is negligible.

In terms of sample complexity, although computing the gradient estimator requires samples, in practice, however, we can reuse the samples that collected and used by the MBRL algorithm, which means we take almost no extra samples to compute the gradient w.r.t task parameters.

Our code is available at `https://github.com/LinZichuan/AdMRL`.

C. Experiments Setup

Low-dimensional velocity-control tasks. Following and extending the setup of (Finn et al., 2017; Rakelly et al., 2019), we first consider a family of environments and tasks relating to controlling 2-D or 3-D velocity control tasks. We consider three popular MuJoCo environments: `Hopper`, `Walker` and `Ant`. For the 3-D task families, we have three task parameters $\psi = (\psi_x, \psi_y, \psi_z)$ which corresponds to the target x-velocity, y-velocity, and z-position. Given the task parameter, the agent's goal is to match the target x and y velocities and z position as much as possible. The reward is defined as: $r_\psi(v_x, v_y, z) = c_1|v_x - \psi_x| + c_2|v_y - \psi_y| + c_3|h_z - \psi_z|$, where v_x and v_y denotes x and y velocities and h_z denotes z height, and c_1, c_2, c_3 are handcrafted coefficients ensuring that each reward component contributes similarly. The set of task parameters ψ is a 3-D box Ψ, which can depend on the particular environment. E.g., `Ant3D` has $\Psi = [-3, 3] \times [-3, 3] \times [0.4, 0.6]$ and here the range for z-position is chosen so that the target can be mostly achievable. For a 2-D task, the setup is similar except only two of these three values are targeted. We experiment with `Hopper2D`, `Walker2D` and `Ant2D`. Details are given in Appendix D. We note that we extend the 2-D settings in (Finn et al., 2017; Rakelly et al., 2019) to 3-D because when the tasks parameters have more degrees of freedom, the task distribution shifts become more prominent.

High-dimensional tasks. We also create a more complex family of high-dimensional tasks to test the strength of our algorithm in dealing with adversarial tasks among a large family of tasks with more degrees of freedom. Specifically, the reward function is linear in the post-transition state s', parameterized by task parameter $\psi \in \mathbb{R}^d$ (where d is the state dimension): $r_\psi(s, a, s') = \psi^\top s'$. Here the task parameter set is $\Psi = [-1, 1]^d$. In other words, the agent's

goal is to take action to make s' most linearly correlated with some target vector ψ. We use `HalfCheetah` where $d = 18$. Note that to ensure that each state coordinate contributes similar to the total reward, we normalize the states by $\frac{s-\mu}{\sigma}$ before computing the reward function, where $\mu, \sigma \in \mathbb{R}^d$ are computed from all states collected by random policy from real environments. The high-dimensional task is called `Cheetah-Highdim` tasks. Tasks parameterized in this way are surprisingly often semantically meaningful, corresponding to rotations, jumping, etc. Appendix F shows some visualization of the trajectories.

Training. We compare our approach with previous meta-RL methods, including MAML (Finn et al., 2017) and PEARL (Rakelly et al., 2019). The training process for our algorithm is outlined in Algorithm 1. We build our algorithm based on the code that (Luo et al., 2018) provides. We use the publicly available code for our baselines MAML, PEARL. Most hyper-parameters are taken directly from the supplied implementation. We list all the hyper-parameters used for all algorithms in the Appendix D. We note here that we only run our algorithm for $n_{tasks} = 10$ or $n_{tasks} = 20$ training tasks, whereas we allow MAML and PEARL to visit 150 tasks during the meta-training for generosity of comparison. The training process of MAML and PEARL requires 80 and 2.5 million samples respectively, while our method AdMRL only requires 0.4 or 0.8 million samples.

Evaluation Metric. For low-dimensional tasks, we enumerate tasks in a grid. For each 2-D environment (`Hopper2D`, `Walker2D`, `Ant2D`) we evaluate at a grid of size 6×6. For the 3-D tasks (`Ant3D`), we evaluate at a box of size $4 \times 4 \times 3$. For high-dimensional tasks, we randomly sample 20 testing tasks uniformly on the boundary. For each task ψ, we compare different algorithms in: $A_0(\psi)$ (zero-shot adaptation performance with no samples), $A_n(\psi)$ (adaptation performance after collecting n samples) and $G_n(\psi) \triangleq A^\star(\psi) - A_n(\psi)$ (suboptimality gap), and $G_n^{\max} = \max_{\psi \in \Psi} G_n(\psi)$ (worst-case suboptimality gap). In our experiments, we compare AdMRL with MAML and PEARL in all environments with $n = 2000, 4000, 6000$. We also compare AdMRL with distributional variants (e.g., model-based methods with uniform or gaussian task sampling distribution) in worst-case tasks, high-dimensional tasks and out-of-distribution (OOD) tasks.

D. Hyper-parameters

We experimented with the following task settings: `Hopper-2D` with x velocity and z height from $\Psi = [-2, 2] \times [1.2, 2.0]$, `Walker-2D` with x velocity and z height from $\Psi = [-2, 2] \times [1.0, 1.8]$, `Ant-2D` with x velocity and y velocity from $\Psi = [-3, 3] \times [-3, 3]$, `Ant-3D` with x velocity, y velocity and z height from $\Psi = [-3, 3] \times [-3, 3] \times [0.4, 0.6]$, `Cheetah-Highdim`
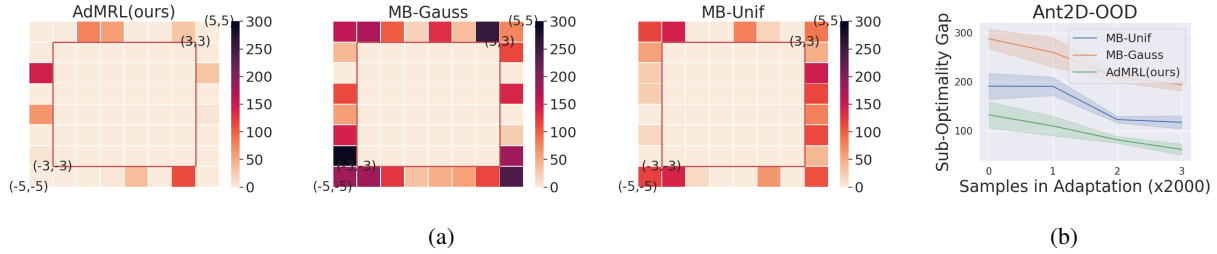
Figure 5. (a) Sub-optimality gap $G_n(\psi)$ of adapted policies $n = 6K$ for each OOD test task ψ of adapted policies from AdMRL, MB-Unif and MB-Gauss. Lighter means smaller, which is better. Training tasks are drawn from $[-3, 3]^2$ (as shown in the red box) while we only test the OOD tasks drawn from $[-5, 5]^2$ (on the boundary). Our approach AdMRL generalizes much better and achieves lower $G_n(\psi)$ than MB-Unif and MB-Gauss on OOD tasks. (b) The worst-case sub-optimality gap $G_n^{\max}$ in the number of adaptation samples n.
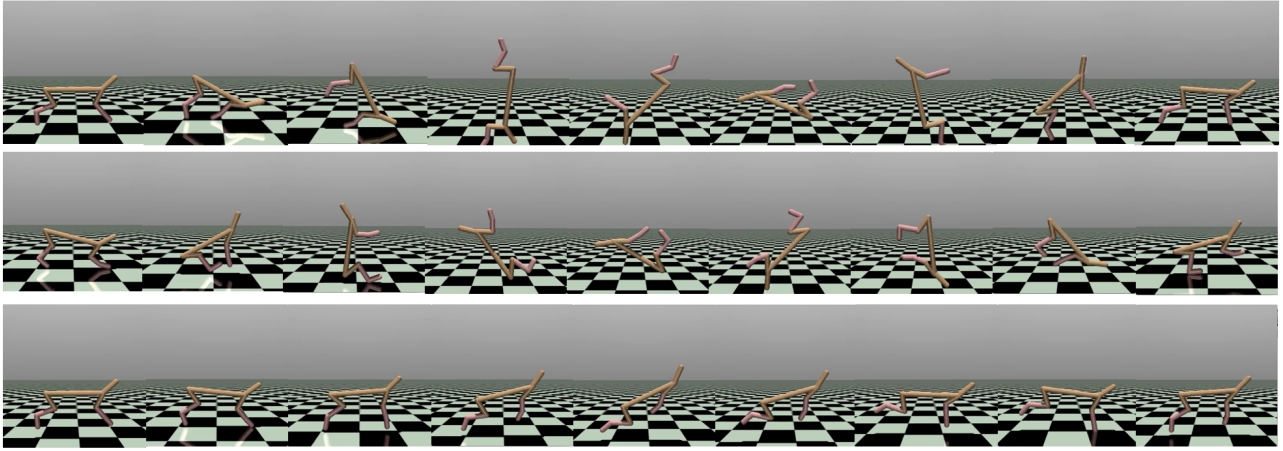

Figure 6. The high-dimensional tasks are surprisingly often semantically meaningful. Policies learned in these tasks can have diverse behaviors, such as front flip (top row), back flip (middle row), jumping (bottom row), etc.

with $\Psi = [-1, 1]^{18}$. We also list the coefficient of the parameterized reward functions in Table 1.

Table 1. Coefficient in parameterized reward functions

	Hopper2D	Walker2D	Ant2D	Ant3D
c_1	1	1	1	1
c_2	0	0	1	1
c_3	5	5	0	30

The hyper-parameters of MAML and PEARL are mostly taken directly from the supplied implementation of (Finn et al., 2017) and (Rakelly et al., 2019). We run MAML for 500 training iterations: for each iteration, MAML uses a meta-batch size of 40 (the number of tasks sampled at each iteration) and a batch size of 20 (the number of rollouts used to compute the policy gradient updates). Overall, MAML requires 80 million samples during meta training. For PEARL, we first collect a batch of training tasks (150) by uniformly sampling from Ψ. We run PEARL for 500 training iterations: for each iteration, PEARL randomly

sample 5 tasks and collects 1000 samples for each task from both prior (400) and posterior (600) of the context variables; for each gradient update, PEARL uses a meta-batch size of 10 and optimizes the parameters of actor, critic and context encoder by 4000 steps of gradient descent. Overall, PEARL requires 2.5 million samples during meta training.

For AdMRL, we first do zero-shot adaptation for each task by 40 virtual steps ($n_{zeroshot} = 40$). We then perform SLBO (Luo et al., 2018) by interleaving data collection, dynamical model fitting and policy updates, where we use 3 outer iterations ($n_{slbo} = 3$) and 20 inner iterations ($n_{inner} = 20$). For each inner iteration, we update model for 100 steps ($n_{model} = 100$), and update policy for 20 steps ($n_{policy} = 20$), each with 10000 virtual samples ($n_{trpo} = 10000$). For the first task, we use $n_{slbo} = 10$ (for `Hopper2D`, `Walker2D`) or $n_{slbo} = 20$ (for `Ant2D`, `Ant3D`, `Cheetah-Highdim`). For all tasks, we sweep the learning rate α in $\{1,2,4,8,16,32\}$ and we use $\alpha = 2$ for `Hopper2D`, $\alpha = 8$ for `Walker2D`, $\alpha = 4$ for `Ant2D` and `Ant3D`, $\alpha = 16$ for `Cheetah-Highdim`.

E. Performance on out-of-distribution tasks

Figure 5 shows the performance of AdMRL on OOD tasks in comparison to MB-Unif and MB-Gauss.

F. Examples of high-dimensional tasks

Figure 6 shows some trajectories in the high-dimensional task `Cheetah-Highdim`.

G. Related Work

The idea of learning to learn was established in a series of previous works (Utgoff, 1986; Schmidhuber, 1987; Thrun, 1996; Thrun & Pratt, 2012). These papers propose to build a base learner for each task and train a meta-learner that learns the shared structure of the base learners and outputs a base learner for a new task. Recent literature mainly instantiates this idea in two directions: (1) learning a meta-learner to predict the base learner (Wang et al., 2016; Snell et al., 2017); (2) learning to update the base learner (Hochreiter et al., 2001; Bengio et al., 1992; Finn et al., 2017). The goal of meta-reinforcement learning is to find a policy that can quickly adapt to new tasks by collecting only a few trajectories. In MAML (Finn et al., 2017), the shared structure learned at train time is a set of policy parameters. Some recent meta-RL algorithms propose to condition the policy on a latent representation of the task (Rakelly et al., 2019; Zintgraf et al., 2019; Wang et al., 2020; Humplik et al., 2019; Lan et al., 2019). Duan et al. (2016); Wang et al. (2016) represent the reinforcement learning algorithm as a recurrent network. Mendonca et al. (2019) improves the sample efficiency during meta-training by consolidating the solutions of individual off-policy learners into a single meta-learner. Schulze et al. meta-learns to perform approximate inference on an unknown task, and incorporate task uncertainty directly during action selection. Rothfuss et al. (2018) improves the sample-efficiency during meta-training by overcoming the issue of poor credit assignment. Some researches (Landolfi et al., 2019; Sæmundsson et al., 2018; Nagabandi et al., 2018a;b) also propose to share a dynamical model across tasks during meta-training and perform model-based adaptation in new tasks. These approaches are still distributional and suffers from distribution shift. We adversarially choose training tasks to address the distribution shift issue and show in the experiment section that we outperform the algorithm with randomly-chosen tasks. Gupta et al. (2018) proposes unsupervised meta-RL, which constructs a task proposal mechanism based on a mutual information objective, to automatically acquire an environment-specific learning procedure. Kirsch et al. (2019) proposes to meta-learn objective functions to generalize to different environments. Mehta et al. (2020) attempts to mitigate the distribution shift issue by introducing a curriculum for meta-training tasks. Fakoor et al. (2019) proposes ways to reuse data from the meta-training phase during meta-adaptation by employing propensity score estimation. Concurrent work by Mendonca et al. (2020) mitigates distribution shift issue by meta-learning a model representation and relabel meta-training experience during adaptation. Different from the method above, our method addresses the distribution shift issue in *task level* by taking a *non-distributional* perspective and meta-training on *adversarial* tasks.

Model-based approaches have long been recognized as a promising avenue for reducing sample complexity of RL algorithms. One popular branch in MBRL is Dyna-style algorithms (Sutton, 1990), which iterates between collecting samples for model update and improving the policy with virtual data generated by the learned model (Luo et al., 2018; Janner et al., 2019; Wang & Ba, 2019; Chua et al., 2018; Buckman et al., 2018; Kurutach et al., 2018; Feinberg et al., 2018; Rajeswaran et al., 2020). Another branch of MBRL produces policies based on *model predictive control* (MPC), where at each time step the model is used to perform planning over a short horizon to select actions (Chua et al., 2018; Nagabandi et al., 2018c; Dong et al., 2019; Wang & Ba, 2019).

Our approach is also related to active learning (Atlas et al., 1990; Lewis & Gale, 1994; Silberman, 1996; Settles, 2009). It aims to find the most useful or difficult data point whereas we are operating in the task space. Our method is also related to curiosity-driven learning (Pathak et al., 2017; Burda et al., 2018a;b), which defines intrinsic curiosity rewards to encourage the agent to explore in an environment. Instead of exploring in state space, our method are "exploring" in the task space. The work of Jin et al. (2020) aims to compute the near-optimal policies for any reward function by sufficient exploration, while we search for the reward function with the worst suboptimality gap.