
A Model-based Approach for Sample-efficient Multi-task Reinforcement Learning

Nicholas C. Landolfi¹ Garrett Thomas¹ Tengyu Ma^{1,2}
{lando, gwthomas, tengyuma}@stanford.edu

¹ Department of Computer Science, Stanford University

² Department of Statistics, Stanford University

Abstract

The aim of multi-task reinforcement learning is two-fold: (1) efficiently learn by training against multiple tasks and (2) quickly adapt, using limited samples, to a variety of new tasks. In this work, the tasks correspond to reward functions for environments with the same (or similar) dynamical models. We propose to learn a dynamical model during the training process and use this model to perform sample-efficient adaptation to new tasks at test time. We use significantly fewer samples by performing policy optimization only in a “virtual” environment whose transitions are given by our learned dynamical model. Our algorithm sequentially trains against several tasks. Upon encountering a new task, we first *warm-up* a policy on our learned dynamical model, which requires no new samples from the environment. We then *adapt the dynamical model* with samples from this policy in the real environment. We evaluate our approach on several continuous control benchmarks and demonstrate its efficacy over MAML, a state-of-the-art meta-learning algorithm, on these tasks.

1 Introduction

Reinforcement learning has achieved significant success in domains such as game-playing [Mnih et al., 2015], recommender systems [Zhang et al., 2019], and robotic control [Levine et al., 2016]. When developing systems that must solve several tasks rather than a single task, it is natural to try to leverage shared structure across tasks to make the learning process more efficient [Caruana, 1997]. Moreover, exploiting shared structure enables faster adaptation to unseen tasks in the future.

The Model Agnostic Meta Learning (MAML) algorithm has proven to be a successful algorithm for multi-task learning in both supervised and reinforcement learning settings [Finn et al., 2017]. In the reinforcement learning (RL) setting, MAML learns a shared policy initialization across tasks. This policy initialization is then adapted to new tasks at test time by policy gradient updates.

Transferring a policy between tasks faces two challenges. First, transferring a policy proves difficult on task distributions in which the optimal policies vary dramatically across tasks. We must assume the existence of a policy initialization from which many task-specific policies may be found via a few local updates. This assumption may fail if policies vary widely across tasks.¹

Second, transferring policies also affects the time and sample efficiency of the adaptation phase. Even if an algorithm exhaustively trains against all tasks in the family, the adaptation phase still requires samples.

¹While Finn and Levine [2017] show that a single step or multiple steps of gradient-based adaptation is powerful and universal in theory, this analysis may require very complex neural networks, which in turn need many samples to generalize.

We aim to address these limitations by revisiting the model-based approach to multi-task reinforcement learning. First, we learn and adapt with a shared dynamical model, rather than a policy. Second, we propose a “warm-up” phase of adaptation in which we train a policy on our learned dynamical model. We assume that the tasks encountered have the same (or similar) associated dynamical models, but we allow the tasks, and the policies required to solve them, to vary arbitrarily. Adaptation with our algorithm requires a modest amount of time (because we train a separate policy for each task), but the adaptation requires few, if any, samples. Consider the extreme case in which we acquire a perfect dynamical model of the environment during training: adapting a policy to a new task may be computationally challenging, but will require no new samples.

Conventional wisdom may suggest the learned dynamical model will fail to transfer; i.e., a dynamical model learned for one task will not work for other tasks. Perhaps surprisingly, we find that the dynamical model can transfer well; especially well if we also collect a small amount of new samples on a task during adaptation. We find our model-based approach provides significant gains in sample complexity over prior state-of-the-art: policies produced by our algorithm obtain better performance on test tasks, despite using less than 1% as much environment interaction at train time (e.g., 0.4 vs. 80 million samples).

In summary, we contribute:

1. A model-based multi-task RL algorithm, Sequential Multi-task Learning (Algorithm 1).
2. Numerical experiments comparing our approach to MAML on several continuous control MuJoCo reinforcement learning benchmarks of varying difficulty (Figure 1).
3. Numerical experiments suggesting our algorithm can handle (a) out-of-distribution tasks, (b) a shift in dynamical model (both Figure 2) and (c) active task selection (Figure 3).

2 Preliminaries

In multi-task learning we train on a variety of tasks to acquire some shared parameters that help us learn new, “similar,” tasks with few samples. In this section we clarify the multi-task RL formalism addressed by our work, review the Model Agnostic Meta Learning (MAML) algorithm in this setting, and recall SLBO, a recent state-of-the-art model-based reinforcement learning algorithm.

2.1 Reinforcement Learning

Consider a Markov Decision Process (MDP) with state space \mathcal{S} and action space \mathcal{A} . The transition dynamics $M(\cdot|s, a)$ specifies the conditional distribution of the next state given the current state s and action a . A reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defines the step-wise reward. Additionally, we fix a discount $\gamma \in [0, 1)$ and an initial state distribution p_0 .

A policy $\pi(\cdot|s)$ specifies a conditional distribution over actions given a state s . We define the value function $V^{\pi, M} : \mathcal{S} \rightarrow \mathbb{R}$ at state s for a policy π and dynamical model M :

$$V^{\pi, M}(s) = \mathbf{E}_{\substack{a_t \sim \pi(\cdot|s_t), \\ s_{t+1} \sim M(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right]. \quad (1)$$

In practice, we truncate the infinite sum to a finite horizon H . The *expected return* $\eta^M(\pi) := \mathbf{E}_{s_0 \sim p_0} [V^{\pi, M}(s_0)]$ gives a one-number summary of a given policy’s performance on an environment with dynamics M . We seek a policy π which maximizes $\eta^M(\pi)$.

2.2 Multi-task Reinforcement Learning

Let $\Psi \subseteq \mathbb{R}^k$ parameterize a family of *tasks* and $\Theta \subseteq \mathbb{R}^p$ parameterize a family of policies. The family of tasks, indexed by ψ , is a family of Markov decision process $\{(\mathcal{S}, \mathcal{A}, M_\psi, r_\psi, p_0, \gamma)\}_{\psi \in \Psi}$. We are primarily interested in tasks that share (or have similar) underlying dynamics and only differ in their reward functions — namely, the transition dynamics $M_\psi = M^*$ for some fixed M^* (or M_ψ ’s are similar). The value function of a policy π on a task with reward r_ψ and dynamics M is denoted $V_\psi^{\pi, M}$, and the expected return $\eta_\psi^M(\pi) = \mathbf{E}[V_\psi^{\pi, M}(s_0)]$ is defined now for each task and dynamical

model. To simplify notation, we will use the shorthand $\eta_\psi^M(\theta) := \eta_\psi^M(\pi_\theta)$, and omit the superscript M in the common case where M is the shared true dynamics M^* .

To reuse knowledge across tasks, algorithms produce a shared *structure*, such as a policy initialization (in MAML) or a dynamical model (in our algorithm). Let $\Phi \subseteq \mathbb{R}^d$ denote the set of all such structures. The training procedure produces one such shared structure $\phi \in \Phi$, which is subsequently used by an adaptation algorithm $A : \Phi \times \Psi \rightarrow \Theta$ at test time to produce policy parameters $\theta = A(\phi, \psi)$ for a given task ψ . We consider a class of adaptation algorithms constrained to a common sample budget, but we suppress this budget in the notation for simplicity.

We aim to find a shared structure ϕ which enables good post-adaptation performance, in expectation, over a given task distribution $p(\psi)$:

$$\underset{\phi}{\text{maximize}} \underset{\psi}{\mathbf{E}} [\eta_\psi(A(\phi, \psi))]. \quad (2)$$

If A is non-deterministic, the expectation above is also taken over the randomness in A .

2.3 Model Agnostic Meta Learning

The celebrated Model Agnostic Meta Learning (MAML) algorithm, as applied to multi-task reinforcement learning, is a model-free policy-search initialization method [Finn et al., 2017]. In MAML, the shared structure learned at train time is a set of policy parameters, i.e., $\Phi = \Theta$. The adaptation algorithm updates the parameters of the policy for a new task so that they will perform well on the new task after a single step of gradient ascent (multiple steps may be used in practice):

$$A_{\text{MAML}}(\theta, \psi) = \theta + \alpha \nabla \eta_\psi(\theta). \quad (3)$$

where $\nabla \eta_\psi(\theta)$ is estimated via policy gradient methods [Williams, 1992, Sutton et al., 2000] and $\alpha \in \mathbb{R}_+$ is a step size. The meta-objective, Eq. 2, is approximately optimized over ϕ by sampling tasks.

2.4 Stochastic Lower Bound Optimization

The recently proposed Stochastic Lower Bound Optimization (SLBO) method is a model-based reinforcement learning algorithm [Luo et al., 2019]. SLBO achieves state-of-the-art sample efficiency by interleaving dynamical model fitting, policy training, and sample collection.

For a learned dynamical model \hat{M}_ϕ , define the k -step prediction \hat{s}_{t+k} on state s_t and action sequence $a_{t:t+k}$ by $\hat{s}_t = s_t$ and $\hat{s}_{t+k+1} = \hat{M}_\phi(\hat{s}_{t+k}, a_{t+k})$ for $k \geq 0$. Define the k -step prediction loss:

$$\mathcal{L}^{(k)}(s_{t:t+k}, a_{t:t+k}; \phi) := \frac{1}{k} \sum_{i=1}^k \|\hat{s}_{t+i} - s_{t+i}\|_2 \quad (4)$$

where we do not square the norm, as in Luo et al. [2019].

Consider a family of policies $\{\pi_\theta\}$ and dynamics models $\{\hat{M}_\phi\}$ (for our purposes, fully connected neural networks). SLBO approximately solves:

$$\underset{\theta, \phi}{\text{maximize}} \left\{ \underbrace{\eta^{\hat{M}_\phi}(\pi_\theta)}_{\text{reward-to-go}} - \lambda \underbrace{\mathbf{E}_{s_{t:t+h}, a_{t:t+h} \sim \pi_{\tilde{\theta}}, M^*} [\mathcal{L}^{(k)}(s_{t:t+k}, a_{t:t+k}; \phi)]}_{\text{transition dynamics fit}} \right\} \quad (5)$$

by alternatively maximizing the reward-to-go on the virtual MDP induced by the learned dynamics \hat{M}_ϕ , and minimizing the transition dynamics fit on data from a policy rollout. Here λ is a hyperparameter which controls how to weight the terms when optimizing and $\tilde{\theta}$ is the policy at the previous iteration. In practice, the transition dynamics are fit with Adam [Kingma and Ba, 2014] and reward-to-go is optimized with TRPO [Schulman et al., 2015].

3 Related Work

A number of previous works have considered multi-task reinforcement learning [Oh et al., 2017, Ammar et al., 2014, Wilson et al., 2007, Lazaric and Ghavamzadeh, 2010], and Taylor and Stone [2009] survey the area. One line of work studies the problem of producing a single policy which can effectively perform a variety of tasks. A common strategy here, often referred to as *distillation*, is to transfer knowledge from one policy to another. The basic approach trains a single policy to imitate many task-specific expert policies [Parisotto et al., 2016, Rusu et al., 2015]. One can additionally regularize the task-specific policies so that they remain somewhat close to the distilled policy [Teh et al., 2017].

Multi-task learning is closely related to *meta-learning*, where a learning algorithm attempts to “learn how to learn”, somehow leveraging prior experience to learn future tasks more quickly [Schmidhuber, 1987, Thrun and Pratt, 1998]. Beyond reinforcement learning, meta-learning has been successfully applied to the problem of few-shot classification, where the goal is to learn (at test time) to accurately classify instances of new classes not encountered during training, given just a few examples from these classes [Santoro et al., 2016, Vinyals et al., 2016, Ravi and Larochelle, 2017]. The MAML algorithm [Finn et al., 2017] approaches meta-learning by learning an initialization that can be quickly adapted to new tasks via gradient-based optimization. Another approach is training a recurrent neural network that is a function of the training history, implicitly encoding a learning algorithm in its weights [Hochreiter et al., 2001]. This strategy has been applied in reinforcement learning by Duan et al. [2016] and Wang et al. [2016]. Clavera et al. [2019] develop both MAML-like and RNN-based meta-learning approaches to learning and adapting dynamical models online in changing environments.

Model-based approaches have long been recognized as a promising avenue for reducing the sample complexity of RL algorithms [Sutton and Barto, 1998, Deisenroth and Rasmussen, 2011]. We use an existing model-based RL algorithm (SLBO), noting that in principle our approach can be used with any model-based RL algorithm, and that SLBO is related to other recently proposed algorithms. The special case where $n_{\text{inner}} = 1$ (i.e. the model is re-trained once each time new data is observed) has been described several times under different names: it is referred to as MB-TRPO in [Luo et al., 2019], as “Vanilla Model-Based Deep Reinforcement Learning” in [Kurutach et al., 2018], and as SimPLe in [Kaiser et al., 2019]. However, it is usually challenging to obtain a perfect dynamics model [Abbeel et al., 2006], and a policy trained on a single estimated model is prone to overfit to particular inaccuracies in that model. Kurutach et al. [2018] propose to use an ensemble of models to mitigate this issue. Using multiple inner iterations in SLBO is similar to using an ensemble; due to stochasticity in the optimizer, the intermediate models obtained after varied numbers of optimization steps are likely to agree on the training data, but may differ outside of the training distribution.

Training a policy in a virtual environment is not the only way to use a learned dynamical model [Sutton and Barto, 1998]. For example, one can produce 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 the next action [Chua et al., 2018]. Nagabandi et al. [2018] use MPC to initialize policies and then fine-tune them using model-free RL. Alternatively, one can use the model to generate “imagined” trajectories and use these as additional inputs to a policy [Racanière et al., 2017].

4 Our Approach

Recall that two areas of interest addressed by our approach are (a) sample complexity in the training time and adaptation time and (b) zero-shot adaptation to similar tasks. To achieve this we leverage two insights:

1. We transfer the parameters of the learned *dynamical model*, rather than the policy.
2. We *warm-up* a task’s policy by training on learned “virtual” dynamics prior to interaction.

Our algorithm proceeds in three iterated stages: (1) task sampling, (2) policy warm-up training, and (3) one or more interleaved data collection and policy training phases. In essence, we perform model-based reinforcement learning on a sequence of tasks sampled from the task distribution $p(\psi)$. We present details on each step below and pseudo-code in Algorithm 1.

4.1 Sequential Multi-task Training Overview

(1) Task Sampling. We could augment this form of sampling by choosing tasks actively, or in some adversarial manner. Later we suggest some heuristics for choosing tasks actively (Section 4.4), and we leave adversarial multi-task setting to later work.

(2) Policy Warm-Up. We initialize a random policy and warm it up on the new task by training on learned dynamical models; for the first task, we skip the warm-up. We call this `VirtualTraining` because it requires no real samples, and trains the policy well against only learned dynamical models of the environment. We intend, of course, that this warm-up procedure on the learned dynamical model will produce a reasonable policy without interacting with the new environment (validated empirically in Section 5).

(3) Data Collection. Here we alternately collect new data, fit a dynamics model and then perform `VirtualTraining` for n_{slbo} iterations. This stage is properly viewed as running SLBO, a model-based RL algorithm, on the new task with a warmed-up policy and previously selected data. Only here (line 7) do we collect samples from the real environment.

Algorithm 1: Sequential Multi-task Training

Result: Model parameters ϕ , dataset \mathcal{D}

```

1 Initialize model parameters  $\phi$  and dataset  $\mathcal{D} \leftarrow \emptyset$ 
2 for  $n_{tasks}$  iterations do
3   Sample task  $\psi \sim p(\psi)$ 
4   Initialize policy parameters  $\theta$ 
5   If  $\mathcal{D} \neq \emptyset$ , VirtualTraining( $\theta, \phi, \mathcal{D}, n_{warmup}$ )
6   (SLBO): for  $n_{slbo}$  iterations do
7      $\mathcal{D} \leftarrow \mathcal{D} \cup \{ \text{collect } n_{collect} \text{ samples from real environment } M \text{ using } \pi_{\theta} \text{ with noise} \}$ 
8     VirtualTraining( $\theta, \phi, \mathcal{D}, n_{inner}$ )
9 subroutine VirtualTraining( $\theta$ : policy,  $\phi$ : model,  $\mathcal{D}$ : data,  $n_{inner}$ ) begin
10   for  $n_{inner}$  iterations do
11     Optimize Eq. 4 over  $\phi$  with data sampled from  $\mathcal{D}$  by  $n_{model}$  steps of Adam
12     for  $n_{policy}$  iterations do
13        $\mathcal{D}' \leftarrow \{ \text{collect } n_{trpo} \text{ samples using learned } \hat{M}_{\phi} \text{ as dynamics} \}$ 
14       Optimize  $\pi_{\theta}$  by running TRPO on  $\mathcal{D}'$ 

```

4.2 Key Design Choices

Inner Iterations of Virtual Training. In steps (2) and (3), we train the policy against several learned dynamical models. By intentional over-parameterization of the neural network representing the dynamical model, we cause the policy to see different dynamical models (all, however consistent with the data) at each iteration of the virtual training process. Intuitively, this process prevents the policy from over-fitting to a particular dynamical model. Empirically, [Luo et al., 2019] found this improved performance in the single-task traditional RL setting.

Warm-up. We warm-up a policy on the trained model. Firstly, this allows us to adapt the policy without any new samples. If we can collect new samples from the environment, this process ensures that we obtain informative data even on the first roll-out. Ultimately when we test our algorithm, our adaptation will be exactly this warm-up phase (followed, perhaps, by a few iterations of data collection and training phases).

Sequential vs. Joint Training. We have chosen to perform training *sequentially*, meaning that we select a task and then train and collect data on it in isolation, before moving to the next task. In contrast, we could instead sample a batch of tasks, then iteratively collect data and train policies for each task *jointly*. The primary advantage of sequential training is that for later tasks, we start with a reasonable policy, and so data collected is very informative. Whereas in joint training, we waste samples early on by collecting data on all tasks when each policy is poor.

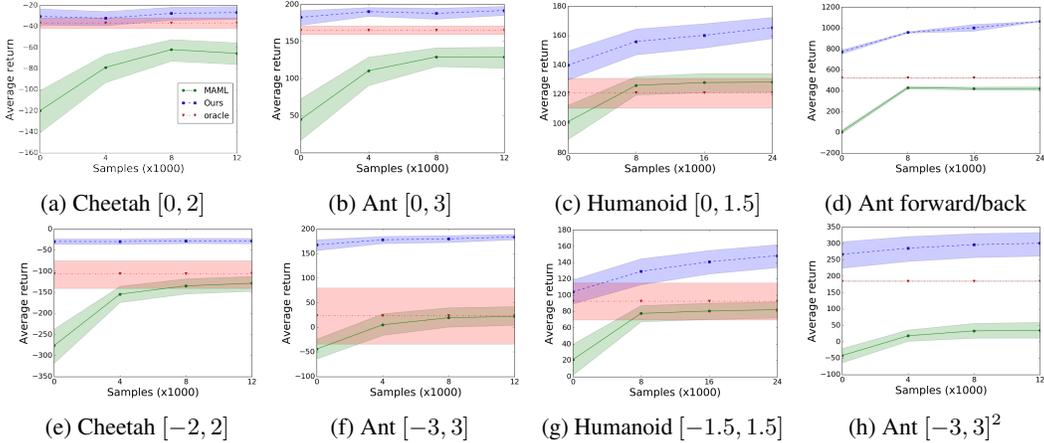


Figure 1: Average return (with 95% confidence interval) across tasks of adapted policies from our algorithm, MAML, and an “oracle” policy trained with access to task parameters as input. We substantially outperform MAML, often with no interaction; a result of transferring a dynamical model and warm-up. See Section 5.2.

4.3 Adaptation / Test Phase

Given a new task, we use the learned model to train a policy from scratch. More precisely, we randomly initialize policy parameters $\tilde{\theta}_0$, warm the policy up using \hat{M}_ϕ , and then continue to adapt with further SLBO training:

$$A_{\text{OURS}}(\phi, \psi) = \text{SLBO}(\text{VT}(\tilde{\theta}_0, \phi, \mathcal{D}, n_{\text{warmup}}), \phi, \mathcal{D}, n_{\text{slbo}}). \quad (6)$$

where VT denotes the `VirtualTraining` routine (lines 9-14 in Algorithm 1) which returns updated policy parameters, and SLBO denotes some number of steps of the SLBO algorithm (lines 6-8 in Algorithm 1).

In Section 5.3 we experimentally analyze our algorithm’s performance when adapting to (a) different task distributions at test time and (b) different dynamical models at test time.

4.4 Active Task Selection

As mentioned earlier, our tasks need not be sampled uniformly. In particular, Line 3 of Algorithm 1 can be extended to select tasks actively. This enables us to select a particularly difficult or diverse sequence of tasks, which may be desirable if it speeds training.

Skipping Tasks. Suppose we have a function $\mu : \Psi \rightarrow \mathbb{R}$ that faithfully rates the difficulty of a task: (ψ_1 harder than ψ_2) $\iff \mu(\psi_1) > \mu(\psi_2)$ where the LHS is domain specific. If we have seen at least L tasks, a straightforward active augmentation to Algorithm 1 at iteration $j > L$ is to skip task ψ_j if $\mu(\psi_j) < \text{quantile}_q(\{\mu(\psi_i)\}_{i=1}^L)$. Intuitively, we want to focus on tasks which are “hard.”

Rating Tasks. One sensible rating function is the difference between a virtual model’s predicted policy reward and the real reward:

$$\mu(\psi_i) = \eta_{\psi_i}^{\hat{M}_\phi}(\theta_i) - \eta_{\psi_i}^{M^*}(\theta_i),$$

where θ_i are parameters of the policy after warm-up. Computing such a rating requires samples, but we can estimate it using several pairs of virtual models. We experimentally explore active task sampling in Section 5.4.

5 Experiments

We evaluate our approach on a variety of continuous control tasks based on environments from the rllab benchmark [Duan et al., 2016], which uses the MuJoCo physics simulator [Todorov et al., 2012]. We first describe the tasks considered, explain the setup of the experiments, present results comparing

our algorithm and MAML. Next, we describe and give results for modified settings where the test tasks are not drawn from the same distribution as the training tasks. Finally, we include preliminary results for methods in which we sample tasks adaptively, rather than independently, at train time.

We implemented our sequential multi-task training algorithm by extending the code that Luo et al. [2019] provide.² We use the MAML implementation that Finn et al. [2017] provide.³ In all cases, we use a horizon of $H = 200$ time steps. Each task involves one of three standard MuJoCo models: *half-cheetah*, *ant*, or *simple humanoid*.

Goal velocity tasks. In *goal velocity* tasks, a velocity ψ is given, and the agent’s goal is to match its own velocity to ψ . In our experiments, ψ is sampled uniformly at random from an interval Ψ . We experimented with the following variants: half-cheetah with $\Psi = [0, 2]$, half-cheetah with $\Psi = [-2, 2]$, ant with $\Psi = [0, 3]$, ant with $\Psi = [-3, 3]$, humanoid with $\Psi = [0, 1.5]$, humanoid with $\psi = [-1.5, 1.5]$, and ant with x and y velocities $\Psi = [-3, 3]^2$.⁴

Forward/backward tasks. In *forward/backward* tasks, a direction $\psi \in \{\pm 1\}$ (where $\psi = 1$ indicates forward and $\psi = -1$ indicates backward) is given, and the agent’s goal is to maximize its speed in the given direction. In our experiments, ψ is sampled uniformly at random from $\Psi = \{\pm 1\}$.

5.1 Training & Evaluation

We compare the performance of our method to MAML in adapting to new tasks sampled from the task distribution. The output of MAML’s meta-training process is a set of initial policy parameters θ_0 , while the output of our method is a set of initial dynamical model parameters ϕ_0 and a dataset \mathcal{D}_0 .

Training. The training process for our algorithm is outlined in Algorithm 1. The training process for MAML is outlined in Finn et al. [2017], Algorithm 1. For both algorithms, and all environments, each policy is implemented as a fully connected feedforward network with two hidden layers of 100 units each, using the ReLU activation $\sigma(x) = \max\{0, x\}$. The sizes of the input and output layers are matched with the dimensions of the state and action spaces, respectively. We list all the hyperparameters used for both algorithms in the Appendix.

We note here that the MAML training process (which we did not modify from Finn et al. [2017]) requires 80 million samples. We ran our algorithm for $n_{\text{tasks}} = 100$ iterations, requiring 0.4 million samples, less than 1% required by MAML⁵.

Evaluation. To evaluate our algorithm against MAML we sample 40 tasks from $p(\psi)$ and train a separate policy for each. For our algorithm we warm-up a policy on the virtual environment ϕ_0 using data sampled from \mathcal{D}_0 , and then train on $n_{\text{itr}} = 3$ stages of SLBO. For MAML, we initialize to the policy θ_0 and then train $n_{\text{itr}} = 3$ policy gradient steps. In both cases, the amount of data collected at each iteration is the same, and is plotted on the horizontal axis of all plots.

5.2 Comparison to MAML

For the eight task/environment pairs described in Section 5, we compare our algorithm against (a) MAML and (b) an *oracle* policy which is trained jointly across the task distribution but receives the task parameter ψ as an additional input (as in [Finn et al., 2017]). We produce the curves in Figure 1 by (1) sampling 40 independently drawn tasks, (2) training a policy by collecting roll-outs (described in Section 5.1), and (3) estimating the policy’s average return by sampling new roll-outs. The plots show the return averaged across tasks, with 95% confidence intervals computed by a bootstrap estimator. Although different tasks have different reward ranges (i.e., the maximum achievable reward depends on the task parameter ψ), the expectation over tasks is comparable. The oracle policy acts roughly as an upper bound of the few-step performance of MAML, but we note that our approach can and often does exceed the performance of the oracle policy, because we train a separate, independent policy for each test task, whereas the oracle policy takes in the task parameters as inputs. MAML

²<https://github.com/roosephu/slbo>

³https://github.com/cbfinn/maml_rl

⁴ $[0, b]$ and forward/backward tasks first appeared in Finn et al. [2017]. We remove contact cost, a negligible part of total reward, as we do not include this in our learned dynamical model.

⁵Ours: 100 tasks by 4000 samples/task. MAML: 500 meta-steps by 40 tasks/meta-step by 4000 samples/tasks.

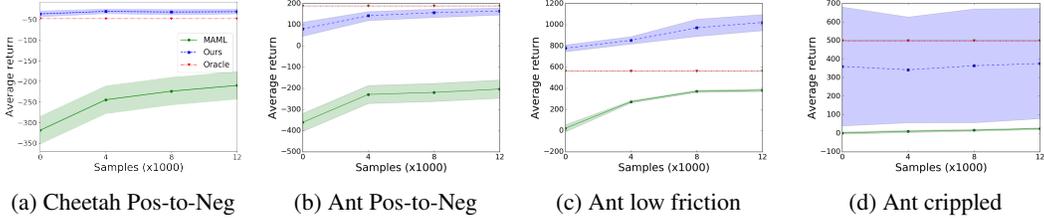


Figure 2: Average return (with 95% confidence interval) across tasks of adapted policies from our algorithm, MAML and the oracle baseline (Section 5.2) for task distribution and dynamical model shift. Even on unseen tasks, warm-up performs well. When underlying dynamical model changes, we (naturally) require some samples to adapt (c) and (d). See Section 5.3 for details.

also trains a policy for each test task, but these are all constrained to have the same initialization, effectively limiting their ability to cover the task distribution.

Analysis: Firstly, we out-perform MAML across all tasks; even having trained with fewer samples. As we mentioned earlier, if the policy must vary substantially on different tasks within the same task family, there is no guarantee that there exists an initialization from which near-optimal policies for all tasks can be reached within one or a few gradient steps. This is a fundamental limitation of MAML which is not shared by our approach, and allows us to substantially out-perform MAML. Secondly, training policies in a virtual environment provided by the learned dynamics model enables *zero-shot adaptation* to new tasks if the model is sufficiently accurate. That is, the policy produced by the warm-up stage (before any samples are collected from the test environment) may already be high-performing. Indeed, we observe our algorithm out-performing the oracle without any samples.

5.3 Task Distribution and Dynamical Model Shift

We also examine how our approach performs when the distribution of tasks at test time differs from that at train time. This scenario is referred to as *domain adaptation* in the context of supervised learning [Ben-David et al., 2010]. We experimented with four such tasks, results in Figure 2.

Changing Reward Distribution. Our first pair of set-ups (one for half-cheetah, and one for ant) evaluate transfer to different reward distributions, without changing the dynamics. In these *positive to negative* tasks, we train on positive velocities ($[0, 2]$ for half-cheetah, $[0, 3]$ for ant) but we test on velocities from the negated interval (i.e., $[-2, 0]$ and $[-3, 0]$, respectively).

Changing True Dynamical Model. Our second two set-ups maintain the same reward distribution, but evaluate transfer to somewhat different underlying dynamical models. In the *ant low friction* scenario, the setup we train on the usual *ant forward/backward* task family, but test in a MuJoCo environment with all friction coefficients halved. In the *ant crippled* scenario, we train on the usual *ant forward/backward* task family, but test in a MuJoCo environment with one ant leg disabled.

Analysis: We see that in the changing reward distributions (Figures 2a, 2b), our algorithm is able to still achieve some gains over MAML as a result of the warm-up phase (compare with Figures 1e and 1f). As expected, if the underlying dynamical model changes (i.e., the friction in Figure 2c) our warm-up is not as effective. Especially so in the case when we cripple the ant, compare the mean reward in Figure 2d (reward ~ 400) with that in Figure 1d (reward ~ 1000).

5.4 Active Task Selection

We validate two instances of the active task selection algorithm described in (Section 4.4): one using the real reward difference rating (requiring extra samples) and one estimating the reward difference (requiring no samples). Figure 3 shows the advantage over the non-active Algorithm 1 on the two-dimensional ant velocity task in three regimes: 5, 20 and 30 tasks.

Analysis: We see that active sampling can not help much with only 5 tasks, helps a modest amount with 20 tasks, and by 30 tasks, our baseline is already doing so well that the active sampling no longer helps. These results suggest the promise of active sampling, but also suggest that the tasks considered in this work are too easy to see any significant gains; our non-active algorithm works well already. We leave it to future work to explore harder tasks and different rating functions.

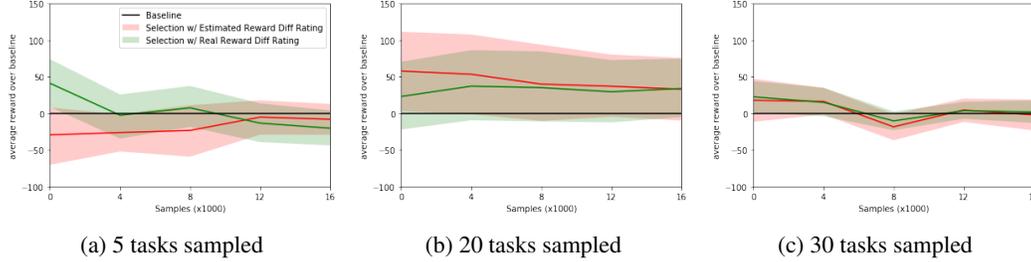


Figure 3: Average difference in return (with 95% confidence intervals) of our active algorithm vs. our non-active algorithm as a baseline. For *ant velocity* $[-3, 3]^2$ environment. We show results for two task difficulty rating functions: estimated (no new samples) and true predicted virtual vs. real policy reward. Active sampling helps in regime of 20 samples (positive return difference with baseline). See Section 5.4 for details.

6 Conclusion

Limitations. Of course, our ability to outperform MAML is enabled by the underlying assumption of similar dynamical models across tasks. In many applications, e.g. robotics, this assumption is reasonable. Still, we do not pretend to solve the general problem when the transition dynamics vary significantly among the tasks to be learned. In that case, zero-shot adaptation becomes impossible, but a partial dynamical model may accelerate adaptation. Second, virtual training, although it requires no new samples, is compute-intensive and the warm-up phase proves the longest part of our algorithm. Though the trade-off with samples is favorable: according to our rough measurements, our less sample-intensive training process still requires less wall-time than MAML.

Future directions. Three clear directions lie ahead. (1) Apply Sequential Multi-task Learning to more difficult settings. In fact, we perform well on most setups in Section 5 within 30 tasks. Also, harder settings would allow us to evaluate active task selection algorithms. (2) Sample tasks *actively* from the distribution. We sketched one possibility in Section 5.4, but upon reaching a sample budget of 30 we observe vanishing gains from active sampling. The questions of rating and comparing tasks, especially with no new samples, remain open. (3) Analyze the online setting and adversarial tasks. Our algorithm handles tasks *sequentially* so the tasks need not come from any distribution. Analysing worst case adaptation performance would be interesting.

Summary. Our model-based approach to multi-task reinforcement learning (a) delivers sample efficiency and (b) enables zero-shot adaptation. We transfer the shared structure of a learned dynamical model and adapt to a new task by training on this learned dynamical model. Our empirical results support the effectiveness of model-based multi-task reinforcement learning.

References

- Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8. ACM, 2006.
- Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In *International Conference on Machine Learning*, pages 1206–1214, 2014.
- Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.
- Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.
- Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.
- Ignasi Clavera, Anusha Nagabandi, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=HyztsoC5Y7>.
- Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.
- Yan Duan, Xi Chen, Rein Houthoofd, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.
- Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL²: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv e-prints*, art. arXiv:1611.02779, Nov 2016.
- Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *arXiv preprint arXiv:1710.11622*, 2017.
- Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.
- Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.
- Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *arXiv e-prints*, art. arXiv:1903.00374, Mar 2019.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=SJJinbWRZ>.
- Alessandro Lazaric and Mohammad Ghavamzadeh. Bayesian multi-task reinforcement learning. In *ICML-27th International Conference on Machine Learning*, pages 599–606. Omnipress, 2010.
- Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

- Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *International Conference on Learning Representations*, volume 7, 2019.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Belle-mare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL <http://dx.doi.org/10.1038/nature14236>.
- Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE International Conference on Robotics and Automation*, pages 7559–7566. IEEE, 2018.
- Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2661–2670. JMLR. org, 2017.
- Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *International Conference on Learning Representations*, 2016.
- Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in neural information processing systems*, pages 5690–5701, 2017.
- Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations*, 2017.
- Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation. *arXiv e-prints*, art. arXiv:1511.06295, Nov 2015.
- Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*, pages 1842–1850, 2016.
- Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.
- John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.
- Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 1998.
- Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.
- Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.
- Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506, 2017.
- Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 1998.
- Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.
- Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

- Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dhharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv e-prints*, art. arXiv:1611.05763, Nov 2016.
- Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
- Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022. ACM, 2007.
- Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):5, 2019.

7 Appendix

7.1 Hyperparameters

For the purpose of reproducibility, we list here all the hyperparameters used in our experiments.

We first describe the settings used for our algorithm. We parameterized the dynamical model by a simple feedforward neural network with two hidden layers of width 500. We used Adam to optimize the reconstruction loss, with a learning rate of 0.001 and a training batch size of 128. We parameterized the policy by a simple feed-forward neural network with two hidden layers of width 32. We used TRPO to optimize the policy with standard parameters from Luo et al. [2019]. As mentioned earlier, most experiments (except notably the active task selection ones) trained against 100 tasks (i.e., $n_{\text{tasks}} = 100$). Additionally, $n_{\text{warmup}} = 40$, $n_{\text{model}} = 100$, $n_{\text{policy}} = 40$ and $n_{\text{slbo}} = 1$ (i.e., we only collected samples once for each task after warm-up). For the adaptation phase, of course, $n_{\text{slbo}} = 4$. On half cheetah environment we collected 4000 samples from the real environment per iteration (with a horizon of 200, this corresponds to 20 trajectories on average). For ant and humanoid we collected 8000 samples from the real environment per iteration (with a horizon of 200, this corresponds to 40 trajectories on average). We always collected the same number of samples when performing TRPO on the virtual dynamics (i.e., $n_{\text{trpo}} = 4000$ or 8000 respectively). We collect the same number of samples at adaptation time.

We now describe the settings used when evaluating MAML. They were mostly taken directly from Finn et al. [2017] or their supplied implementation. The adaptation steps are computed by standard policy gradients [Williams, 1992], while the meta-updates are computed by TRPO [Schulman et al., 2015]. We performed 500 iterations of MAML training. The learning rate (referred to as α in Finn et al. [2017]) is set to 0.1 during the meta-train phase; during the meta-test phase, $\alpha = 0.1$ for the first step and $\alpha = 0.05$ for subsequent steps. The meta-learning rate (referred to as β in Finn et al. [2017]) is set to 0.01. The batch size (the number of rollouts used to compute the policy gradient updates) is 20, except for the ant forward/backward and all humanoid tasks, where we use 40. The meta-batch size (the number of tasks sampled at each iteration of MAML) is 40 in all cases.