Meta-Reinforcement Learning by Tracking Task Non-stationarity

Riccardo Poiani[1*] , **Andrea Tirinzoni**[2†] and **Marcello Restelli**[1]

[1]Politecnico di Milano
[2]Inria Lille

riccardo.poiani@mail.polimi.it, andrea.tirinzoni@inria.fr, marcello.restelli@polimi.it

Abstract

Many real-world domains are subject to a structured non-stationarity which affects the agent's goals and the environmental dynamics. Meta-reinforcement learning (RL) has been shown successful for training agents that quickly adapt to related tasks. However, most of the existing meta-RL algorithms for non-stationary domains either make strong assumptions on the task generation process or require sampling from it at training time. In this paper, we propose a novel algorithm (TRIO) that optimizes for the future by explicitly tracking the task evolution through time. At training time, TRIO learns a variational module to quickly identify latent parameters from experience samples. This module is learned jointly with an optimal exploration policy that takes task uncertainty into account. At test time, TRIO tracks the evolution of the latent parameters online, hence reducing the uncertainty over future tasks and obtaining fast adaptation through the meta-learned policy. Unlike most existing methods, TRIO does not assume Markovian task-evolution processes, it does not require information about the non-stationarity at training time, and it captures complex changes undergoing in the environment. We evaluate our algorithm on different simulated problems and show it outperforms competitive baselines.

1 Introduction

The ability to generalize and quickly adapt to non-stationary environments, where the dynamics and rewards might change through time, is a key component towards building lifelong reinforcement learning (RL) [Sutton and Barto, 2018] agents. In real domains, the evolution of these environments is often governed by underlying structural and *temporal patterns*. Consider, for instance, a mobile robot navigating an outdoor environment where the terrain conditions are subject to seasonal evolution due to climate change; or where the robot's

---
*Contact Author
†Work done while at Politecnico di Milano.

actuators become less effective over time, e.g., due to the natural degradation of its joints or to the system running out of power; or where the designer changes its desiderata, e.g., how the robot should trade off high-speed movement and safe navigation. The commonality is some unobserved *latent variable* (e.g., the terrain condition, the joints' friction, etc.) that evolves over time with some *unknown* temporal pattern (e.g., a cyclic or smooth change). In this setting, we would expect a good RL agent to (1) quickly adapt to different realizable tasks and (2) to extrapolate and exploit the temporal structure so as to reduce the uncertainty over, and thus further accelerate adaptation to, future tasks.

Meta-RL has proven a powerful methodology for training agents that quickly adapt to related tasks [Duan *et al.*, 2016; Wang *et al.*, 2016; Finn *et al.*, 2017; Hospedales *et al.*, 2020]. The common assumption is that tasks are i.i.d. from some unknown distribution from which the agent can sample at training time. This assumption is clearly violated in the lifelong/non-stationary setting, where tasks are temporally correlated. Some attempts have been made to extend meta-RL algorithms to deal with temporally-correlated tasks [Al-Shedivat *et al.*, 2018; Nagabandi *et al.*, 2018; Clavera *et al.*, 2019; Kaushik *et al.*, 2020; Kamienny *et al.*, 2020; Xie *et al.*, 2020]. However, current methods to tackle this problem have limitations. Some of them [Al-Shedivat *et al.*, 2018; Xie *et al.*, 2020] model the task evolution as a Markov chain (i.e., the distribution of the next task depends only on the current one). While this allows capturing some cyclic patterns (like seasonal climate change), it is unable to capture more complex behaviors that are frequent in the real world [Padakandla, 2020]. Other works [Kamienny *et al.*, 2020] consider history-dependent task-evolution processes but assume the possibility of sampling them during the training. While this assumption seems more reasonable for cyclic processes, where the agent experiences a "cycle" infinite many times, it is difficult to imagine that the agent could sample from the same non-stationary process it will face once deployed. Finally, some works [Nagabandi *et al.*, 2018; Clavera *et al.*, 2019; Kaushik *et al.*, 2020] do not explicitly model the task evolution and only meta-learn a policy for fast adaptation to changes. This makes it difficult to handle task-evolution processes other than what they are trained for. These limitations raise our main question: how can we build agents that are able to extrapolate and exploit complex

history-dependent task-evolution processes at test time without prior knowledge about them at training time?

In this paper, we consider the common piecewise stationary setting [Chandak *et al.*, 2020; Xie *et al.*, 2020], where the task remains fixed for a certain number of steps after which it may change. Each task is characterized by an unobserved latent vector of parameters which evolves according to an unknown history-dependent stochastic process. We propose a novel algorithm named TRIO (TRacking, Inference, and policy Optimization) that is meta-trained on tasks from the given family drawn from different prior distributions, while inferring the "right" prior distribution on future tasks by tracking the non-stationarity entirely at test time. More precisely, TRIO meta-trains (1) a variational module to quickly infer a distribution over latent parameters from experience samples of tasks in the given family, and (2) a policy that trades off exploration and exploitation given the task uncertainty produced by the inference module. At test time, TRIO uses curve fitting to track the evolution of the latent parameters. This allows computing a prior distribution over future tasks, thus improving the inference from the variational module and the fast adaptation of the meta-learned policy. We report experiments on different domains which confirm that TRIO successfully adapts to different unknown non-stationarities at test time, achieving better performance than competitive baselines. [1]

2 Preliminaries

We model each task as a Markov decision process (MDP) [Puterman, 1994] $\mathcal{M}_\omega = (\mathcal{S}, \mathcal{A}, \mathcal{R}_\omega, \mathcal{P}_\omega, p_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}_\omega : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $\mathcal{P}_\omega : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state-transition probability function, p_0 is the initial state distribution, and $\gamma \in [0, 1]$ is the discount factor. We assume each task to be described by a latent vector of parameters $\omega \in \Omega \subset \mathbb{R}^d$ that governs the rewards and the dynamics of the environment, and we denote by $\mathfrak{M} := \{\mathcal{M}_\omega : \omega \in \Omega\}$ the family of MDPs with this parameterization (i.e., the set of possible tasks that the agent can face). We consider episodic interactions with a sequence of MDPs $\mathcal{M}_{\omega_0}, \mathcal{M}_{\omega_1}, \ldots$ from the given family $\mathfrak{M}$, which, as in the common piece-wise stationary setting [Xie *et al.*, 2020], remain fixed within an episode. The evolution of these tasks (equivalently, of their parameters) is governed by a history-dependent stochastic process ρ, such that $\omega_t \sim \rho(\omega_0, \ldots, \omega_{t-1})$. The agent interacts with each MDP $\mathcal{M}_{\omega_t}$ for one episode, after which the task changes according to ρ. At the beginning of the t-th episode, an initial state $s_{t,0}$ is drawn from p_0; then, the agent chooses an action $a_{t,0}$, it transitions to a new state $s_{t,1} \sim \mathcal{P}_{\omega_t}(s_{t,0}, a_{t,0})$, it receives a reward $r_{t,1} = \mathcal{R}_{\omega_t}(s_{t,0}, a_{t,0}, s_{t,1})$, and the whole process is repeated for H_t steps.[2] The agent chooses its actions by means of a possibly history-dependent policy, $a_{t,h} \sim \pi(\tau_{t,h})$, with $\tau_{t,h} := (s_{t,0}, a_{t,0}, s_{t,1}, r_{t,1}, \ldots, s_{t,h})$ denoting a h-step trajectory, and the goal is to find a policy

that maximizes the expected cumulative reward across the sequence of tasks,

$$\underset{\pi}{\arg\max} \, \mathbb{E}_{\omega_t \sim \rho} \left[\sum_{t=0}^{T-1} \mathbb{E} \left[\sum_{h=0}^{H_t-1} \gamma^h r_{t,h} \, \big| \mathcal{M}_{\omega_t}, \pi \right] \right]. \quad (1)$$

This setting is conceptually similar to hidden-parameter MDPs [Doshi-Velez and Konidaris, 2013], which have been used to model non-stationarity [Xie *et al.*, 2020], with the difference that we allow complex history-dependent task-generation processes instead of i.i.d. or Markov distributions.

Meta-learning setup. As usual, two phases take place. In the first phase, called *meta-training*, the agent is trained to solve tasks from the given family $\mathfrak{M}$. In the second phase, namely *meta-testing*, the agent is deployed and its performance is evaluated on a sequence of tasks drawn from ρ. As in [Humplik *et al.*, 2019; Kamienny *et al.*, 2020], we assume that the agent has access to the descriptor ω of the tasks it faces during training, while this information is not available at test time. More precisely, we suppose that the agent can train on any task $\mathcal{M}_\omega$ (for a chosen parameter ω) in the family $\mathfrak{M}$. These assumptions are motivated by the fact that, in practical applications, the task distribution for meta-training is often under the designer's control [Humplik *et al.*, 2019]. Furthermore, unlike existing works, we assume that the agent has *no knowledge* about the sequence of tasks it will face at test time, i.e., about the generation process ρ. This introduces the main challenge, and novelty, of this work: how to extrapolate useful information from the family of tasks $\mathfrak{M}$ at training time so as to build agents that successfully adapt to unknown sequences of tasks at test time.

3 Method

Imagine to have an oracle that, only at test time, provides the distribution of the parameters of each task *before* actually interacting with it (i.e., that provides $\rho(\omega_1, \ldots, \omega_t)$ before episode $t+1$ begins). How could we exploit this knowledge? Clearly, it would be of little use without an agent that knows how the latent parameters affect the underlying environment and/or how to translate this uncertainty into optimal behavior. Furthermore, this oracle works only at test time, so we cannot meta-train an agent with these capabilities using such information. The basic idea behind TRIO is that, although we cannot train on the actual non-stationarity ρ, it is possible to prepare the agent to face different levels of task uncertainty (namely different *prior* distributions generating ω) by interacting with the given family $\mathfrak{M}$, so as to adapt to the actual process provided by the oracle at test time. More precisely, TRIO simulates possible priors from a family of distributions $p_z(\omega) = p(\omega|z)$ parameterized by z and practices on tasks drawn from them. Then, TRIO meta learns two components. The first is a module that infers latent variables from observed trajectories, namely that approximates the posterior distribution $p(\omega|\tau, z)$ of the parameters ω under the prior p_z given a trajectory τ. Second, it meta-learns a policy to perform optimally under tasks with different uncertainty. A particular choice for this policy is a model whose input is augmented with the posterior distribution over parameters computed by the inference module. This resembles

---

[1]An extended version of the paper with appendix is available on arXiv.

[2]The length H_t of the t-th episode can be a task-dependent random variable (e.g., the time a terminal state is reached).

Algorithm 1 TRIO (meta-training)

Require: Task family $\mathfrak{M}$, hyperprior $p(z)$, batch size n
1: Randomly initialize θ and ϕ
2: **while** not done **do**
3: Sample prior parameters $\{z_i\}_{i=1}^n$ from $p(z)$
4: Sample task parameters $\{\omega_i\}_{i=1}^n$ from $\{p_{z_i}(\omega)\}_{i=1}^n$
5: Collect $\{\tau_i\}_{i=1}^n$ using policy π_θ in MDPs $\{\mathcal{M}_{\omega_i}\}_{i=1}^n$
6: Update θ by optimizing (4) using $\{\tau_i\}_{i=1}^n$
7: Update ϕ by optimizing (3) using $\{z_i, \omega_i, \tau_i\}_{i=1}^n$
8: **end while**
Ensure: Meta-policy π_θ and inference network q_ϕ

Algorithm 2 TRIO (meta-testing)

Require: Meta-policy π_θ, inference network q_ϕ, stream of
tasks $\omega_t \sim \rho$, initial prior parameters $\widehat{z}_0$
1: Initialize $D_\omega = \emptyset$
2: **for** $t = 0, 1, \dots$ **do**
3: Interact with $\mathcal{M}_{\omega_t}$ using $\pi_\theta, q_\phi, \widehat{z}_t$ and collect τ_t
4: Predict $\widehat{\omega}_t$ using $q_\phi(\tau_t, \widehat{z}_t)$ and set $D_\omega = D_\omega \cup \{\widehat{\omega}_t\}$
5: Fit Gaussian processes using D_ω and predict $\widehat{z}_{t+1}$
6: **end for**

---

a Bayes-optimal policy and allows trading off between gathering information to reduce task uncertainty and exploiting past knowledge to maximize rewards.

At test time, the two models computed in the training phase can be readily used in combination with ρ (which replaces the simulated priors) to quickly adapt to each task. Obviously, in practice we do not have access to the oracle that we imagined earlier. The second simple intuition behind TRIO is that the process ρ can be tracked entirely at test time by resorting to curve fitting. In fact, after completing the t-th test episode, the inference model outputs an approximate posterior distribution of the latent parameter ω_t. This, in combination with past predictions, can be used to fit a model that approximates the distribution of the latent variables ω_{t+1} at the next episode, which in turn can be used as the new prior for the inference model when learning the future task.

Formally, TRIO meta-trains two modules represented by deep neural networks: (1) an inference model $q_\phi(\tau, z)$, parameterized by ϕ, that approximates the posterior distribution $p(\omega | \tau, z)$, and (2) a policy $\pi_\theta(s, q_\phi)$, parameterized by θ, that chooses actions given states and distributions over latent parameters. At test time, TRIO learns a model $f(t)$ that approximates $\rho(\omega_0, \dots, \omega_{t-1})$, namely the distribution over the t-th latent parameter given the previous ones. We now describe each of these components in detail, while the pseudo-code of TRIO can be found in Algorithm 1 and 2.

3.1 Task Inference

As mentioned before, the inference module aims at approximating the posterior distribution $p(\omega | \tau, z)$ of the latent variable ω given a trajectory τ and the prior's parameter z. Clearly, computing the exact posterior distribution $p(\omega | \tau, z) \propto p(\tau | \omega) p_z(\omega)$ is not possible since the likelihood $p(\tau | \omega)$ depends on the true models of the environ-

ment $\mathcal{P}_\omega$ and $\mathcal{R}_\omega$, which are unknown. Even if these models were known, computing $p(\omega | \tau, z)$ requires marginalizing over the latent space, which would be intractable in most cases of practical interest. A common principled solution is variational inference [Blei *et al.*, 2017], which approximates $p(\omega | \tau, z)$ with a family of tractable distributions. A convenient choice is the family of multivariate Gaussian distributions over the latent space $\mathbb{R}^d$ with independent components (i.e., with diagonal covariance matrix). Suppose that, at training time, we consider priors $p_z(\omega)$ in this family, i.e., we consider $p_z(\omega) = \mathcal{N}(\mu, \Sigma)$ with parameters $z = (\mu, \sigma)$ given by the mean $\mu \in \mathbb{R}^d$ and variance $\sigma \in \mathbb{R}^d$ vectors, which yield covariance $\Sigma = \text{diag}(\sigma)$. Then, we approximate the posterior as $q_\phi(\tau, z) = \mathcal{N}(\mu_\phi(\tau, z), \Sigma_\phi(\tau, z))$, where $\mu_\phi(\tau, z) \in \mathbb{R}^d$ and $\Sigma_\phi(\tau, z) = \text{diag}(\sigma_\phi(\tau, z))$ are the outputs of a recurrent neural network with parameters ϕ.

To train the inference network q_ϕ, we consider a *hyperprior* $p(z)$ over the prior's parameters z and directly minimize the expected Kullback-Leibler (KL) divergence between $q_\phi(\tau, z)$ and the true posterior $p(\omega | \tau, z)$. Using standard tools from variational inference, this can be shown equivalent to minimizing the *evidence lower bound* (ELBO) [Blei *et al.*, 2017],

$$\underset{\phi}{\text{argmin}} \, \mathbb{E}\Big[\mathbb{E}_{\widehat{\omega} \sim q_\phi}[\log p(\tau | \widehat{\omega}, z)] + \text{KL}\big(q_\phi(\tau, z) \big\| p_z\big)\Big], \quad (2)$$

where the outer expectation is under the joint process $p(\tau, \omega, z)$. In practice, this objective can be approximated by Monte Carlo sampling. More precisely, TRIO samples the prior's parameters z from $p(z)$, the latent variable ω from $p_z(\omega)$, and a trajectory τ by interacting with $\mathcal{M}_\omega$ under the current policy. Under a suitable likelihood model, this yields the following objective:

$$\underset{\phi}{\text{argmin}} \sum_{i=1}^n \Big(\|\mu_\phi(\tau_i, z_i) - \omega_i\|^2 + \text{Tr}(\Sigma_\phi(\tau_i, z_i))$$
$$+ \frac{\lambda}{H_i} \text{KL}(q_\phi(\tau_i, z_i) \| p_{z_i})\Big). \quad (3)$$

Here we recognize the contribution of three terms; (1) the first one is the standard mean-square error and requires the mean-function $\mu_\phi(\tau, z)$ to predict well the observed tasks (whose parameter is known at training time); (2) the second term encodes the intuition that this prediction should be the least uncertain possible (i.e., that the variances of each component should be small); (3) the last term forces the approximate posterior to stay close to the prior $p_z(\omega)$, where the closeness is controlled as usual by a tunable parameter $\lambda \geq 0$ and by the length H_i of the i-th trajectory.

3.2 Policy Optimization

The agent's policy aims at properly trading off exploration and exploitation under uncertainty on the task's latent parameters ω. In principle, any technique that leverages a given distribution over the latent variables can be used for this purpose. Here we describe two convenient choices.

Bayes-optimal policy. Similarly to [Zintgraf *et al.*, 2019], we model the policy as a deep neural network $\pi_\theta(s, z)$, parametrized by θ, which, given an environment state s and a

Gaussian distribution over the task's parameters, produces a distributions over actions. The former distribution is encoded by the vector $z = (\mu, \sigma)$ which is obtained from the prior and refined by the inference network as the data is collected. This policy is meta-trained to directly maximize rewards on the observed trajectories by proximal policy optimization (PPO) [Schulman *et al.*, 2017],

$$\underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{h=0}^{H_i-1} \gamma^h r_{h,i}, \qquad (4)$$

where the sum is over samples obtained through the same process as for the inference module. Similarly to the inference network, the policy is meta-tested without further adaption. Intuitively, being provided with the belief about the task under consideration, this "Bayes-optimal" policy automatically trades off between taking actions that allow it to quickly infer the latent parameters (i.e., those that are favorable to the inference network) and taking actions that yield high rewards.

Thompson sampling. Instead of using an uncertainty-aware model, we simply optimize a task-conditioned policy $\pi_\theta(s, \omega)$ to maximize rewards (recall that we have access to ω at training time). That is, we seek a *multi-task* policy, perhaps one of the most common models adopted in the related literature [Lan *et al.*, 2019; Humplik *et al.*, 2019]. Then, at test time, we can use this policy in combination with Thompson sampling [Thompson, 1933] (a.k.a. posterior sampling) to trade-off exploration and exploitation in a principled way. That is, we sample some parameter $\omega \sim q_\phi(\tau, z)$ from the posterior computed by the inference network, choose an action according to $\pi_\theta(s, \omega)$, feed the outcome back into the inference network to refine its prediction and repeat this process for the whole episode. As we shall see in our experiments, although simpler than training the Bayes-optimal policy, this approach provides competitive performance in practice.

3.3 Tracking the Latent Variables

As we discussed in the previous sections, before interacting with a given task, both the inference network $q_\phi(\tau, z)$ and the policy $\pi_\theta(s, z)$ (assuming that we use the Bayes-optimal model) require as input the parameter z of the prior under which the task's latent variables are generated. While at meta-training we explicitly generate these parameters from the hyperprior $p(z)$, at meta-testing we do not have access to this information. A simple workaround would be to use non-informative priors (e.g., a zero-mean Gaussian with large variance). Unfortunately, this would completely ignore that, at test-time, tasks are sequentially correlated through the unknown process ρ. Therefore, we decide to *track* this process online, so that at each episode t we can predict the distribution of the next task in terms of its parameter $\widehat{z}_{t+1}$. While many techniques (e.g., for time-series analysis) could be adopted to this purpose, we decide to model ρ as a Gaussian process (GP) [Rasmussen, 2003] due to its flexibility and ability to compute prediction uncertainty. Formally, at the end of the t-th episode, we have access to estimates $\widehat{\omega}_0, \ldots, \widehat{\omega}_t$ of the past latent parameters obtained through the inference network *after* facing the corresponding tasks. We use these data to fit a separate GP for each of the d dimensions of

the latent variables, while using its prediction one-step ahead $\widehat{z}_{t+1} = (\widehat{\mu}_{t+1}, \widehat{\sigma}_{t+1})$ as the prior for the next episode. Intuitively, when ρ is properly tracked, this reduces the uncertainty over future tasks, hence improving both inference and exploration in future episodes.

4 Related Works

Meta-reinforcement learning. The earliest approaches to meta-RL make use of *recurrent networks* to aggregate past experience so as to build an internal representation that helps the agent adapt to multiple tasks [Hochreiter *et al.*, 2001; Wang *et al.*, 2016; Duan *et al.*, 2016]. *Gradient-based* methods, on the other hand, learn a model initialization that can be adapted to new tasks with only a few gradient steps at test time [Finn *et al.*, 2017; Rothfuss *et al.*, 2019]. Some approaches of this kind have been used to tackle dynamic scenarios [Nagabandi *et al.*, 2018; Clavera *et al.*, 2019; Kaushik *et al.*, 2020]. [Al-Shedivat *et al.*, 2018] use few-shot gradient-based methods to adapt to sequences of tasks. Unlike our work, they handle only Markovian task evolution processes and use the knowledge of non-stationarity at training time. Another line of work, which has recently gained considerable attention, considers *context-based* methods that directly take the task uncertainty into account by building and inferring latent representations of the environment. [Rakelly *et al.*, 2019] propose an off-policy algorithm that meta-trains two modules: a variational autoencoder that builds a latent representation of the task the agent is facing, and a task-conditioned optimal policy that, in combination with posterior sampling, enables structured exploration of new tasks. [Zintgraf *et al.*, 2019] design a similar model, with the main difference that the policy is conditioned on the entire posterior distribution over tasks, thus approximating a Bayes-optimal policy. All of these methods are mainly designed for stationary multi-task settings, while our focus is on non-stationary environments. For the latter setup, [Kamienny *et al.*, 2020] meta-learn a reward-driven representation of the latent space that is used to condition an optimal policy. Compared to our work, they deal with continuously-changing environments and assume the possibility of "simulating" this non-stationarity at training time, an assumption that might be violated in many real settings.

Non-stationary reinforcement learning. Since most real-world applications involve environments that change over time, non-stationary reinforcement learning is constantly gaining attention in the literature (see [Padakandla, 2020] for a detailed survey). [Xie *et al.*, 2020] aim at learning dynamics associated with the latent task parameters and perform online inference of these factors. However, their model is limited by the assumption of Markovian inter-task dynamics. Similar ideas can be found in [Chandak *et al.*, 2020], where the authors perform curve fitting to predict the agent's return on future tasks so as to prepare their policy for changes in the environment. Here, instead, we use curve fitting to track the evolution of the latent task parameters and we learn a policy conditioned on them.

5 Experiments

Our experiments aim at addressing the following questions:

- Does TRIO successfully track and anticipate changes in the latent variables governing the problem? How does it perform under different non-stationarities?

- What is the advantage w.r.t. methods that neglect the non-stationarity? How better an oracle that knows the task evolution process can be?

To this end, we evaluate the performances of TRIO in comparison with the following baselines:

- **Oracles**. At the beginning of each episode, they have access to the correct prior from which the current task is sampled. They represent the best that the proposed method can achieve.

- **VariBAD** [Zintgraf *et al.*, 2019] and **RL2** [Wang *et al.*, 2016], which allow us to evaluate the gain of tracking non-stationary evolutions w.r.t. inferring the current task from scratch at the beginning of each episode.

- **MAML (Oracle)** [Finn *et al.*, 2017]. To evaluate gradient-based methods for non-stationary settings, we report the "oracle" performance of MAML *post-adaptation* (i.e., after observing multiple rollouts from the current task).

Furthermore, we test two versions of our approach: Bayes-TRIO, where the algorithm uses the Bayes-optimal policy model, and TS-TRIO, where we use a multi-task policy in combination with Thompson sampling. Additional details and further results, can be found in the supplementary material.

5.1 Minigolf

In our first experimental domain, we consider an agent who is playing a minigolf game day after day. In the minigolf domain [Tirinzoni *et al.*, 2019], the agent has to shoot a ball, which moves along a level surface, inside a hole with the minimum number of strokes. In particular, given the position of the ball on the track, the agent directly controls the angular velocity of the putter. The reward is 0 when the ball enters the hole, -100 if it goes beyond the hole, and -1 otherwise. The problem is non-stationary due to different weather conditions affecting the dynamic friction coefficient of the ground. This, in turn, greatly affects the optimal behavior. Information on how this coefficient changes are unknown a-priori, thus they cannot be used at training time. However, the temporal structure of these changes make them suitable for tracking online. At test time, we consider two realistic models of the ground's friction non-stationarity: A) a sinusoidal function, which models the possible seasonal behavior due to changing weather conditions; and B) a sawtooth-shaped function, which models a golf course whose conditions deteriorate over time and that is periodically restored by human operators when the quality level drops below a certain threshold.

Let us first analyze the tracking of the latent variables in Figure 1 (*bottom*). As we can see, the proposed algorithms are able to successfully track the a-priori unknown evolution of the friction coefficient in both sequences.

As shown in Figure 1 *(top)*, Bayes-TRIO achieves the best results in this domain. It is worth noting that its performance overlaps with the one of its oracle variant for the whole sinusoidal task sequence, while in the sawtooth ones performance drops occur only when the golf course gets repaired (i.e., when its friction changes abruptly). Indeed, before these abrupt changes occur, the agent believes that the next task would have a higher friction and, thus, it strongly shoots the ball towards the goal. However, as the friction abruptly drops, the agent overshoots the hole, thus incurring a highly negative reward. This behavior is avoided in few episodes, when the agent recovers the right evolution of the friction coefficient and restores high performance. A similar reasoning applies to TS-TRIO, which, however, obtains much lower performance, especially in sequence A. The main cause of this problem is its naïve exploration policy and the way TS handles task uncertainty. In fact, since its policy is trained conditioned on the true task, the agent successfully learns how to deal with correct friction values; however, even when negligible errors in the inference procedure are present, the agent incurs catastrophic behaviors when dealing with small friction values and it overshoots the ball beyond the hole. When the friction is greater, as close to the peaks of sequence B, the errors in the inference network have less impact on the resulting optimal policies, and TS-TRIO achieves the same performance as Bayes-TRIO.
VariBAD achieves high performance in situations of low abrasion, but its expected reward decreases as friction increases. This is due to the fact that, at the beginning of each episode, the agent swings the putter softly seeking information about the current abrasion level. While this turns out to be optimal for small frictions, as soon as the abrasion increases, these initial shots become useless: if the agent knew that a high friction is present, it could shoot stronger from the beginning without risking to overshoot the ball. A similar behavior is observed for RL2. Finally, MAML (Oracle) suffers from worse local maxima than context-based approaches and performs significantly worse.

5.2 MuJoCo

We show that TRIO successfully scales to more complex problems by evaluating its performance on two MuJoCo benchmarks typically adopted in the meta-RL literature. We consider two different environments: 1) HalfCheetahVel, in which the agent has to run at a certain target velocity; and 2) AntGoal, where the agent needs to reach a certain goal in the 2D space. We modify the usual HalfCheetahVel reward function to make the problem more challenging as follows: together with a control cost, the agent gets as reward the difference between its velocity and the target velocity; however, when this difference is greater than 0.5, an additional -10 is added to model the danger of high errors in the target velocity and to incentivize the agent to reach an acceptable speed in the smallest amount of time. For AntGoal, we consider the typical reward function composed of a control cost, a contact cost, and the distance to the goal position. At test time, the non-stationarity affects the target speed in HalfCheetahVel and the goal position in AntGoal. We consider different non-linear sequences to show that TRIO can
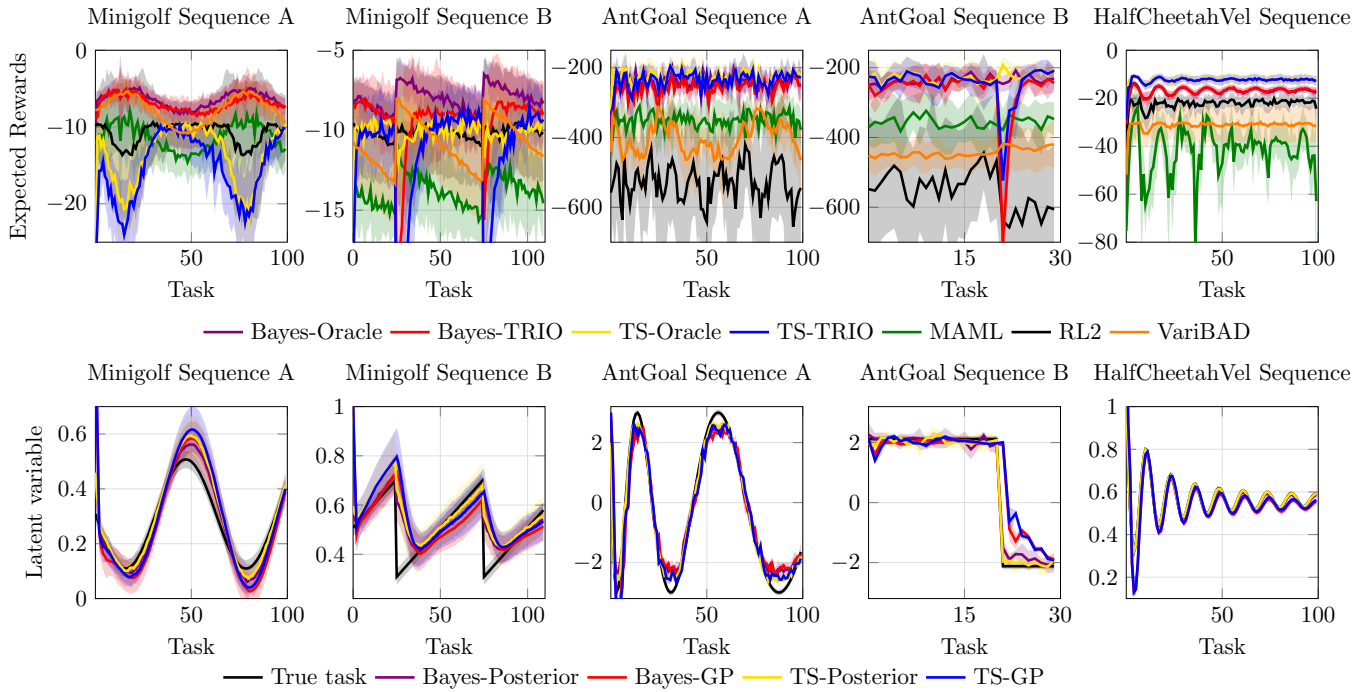
Figure 1: Meta-test performance on different sequences of the selected domains. All plots concerning the Minigolf (MuJoCo) domain are averages and standard deviations of 20 (5) policies, each of which is tested 50 times on the same episode; each task is composed of 4 (1) episodes. *(top)* Expected rewards per task. *(bottom)* Latent-variable tracking per task. The figures report the true latent variable of each task (True task), the posterior mean of TRIO at the end of each task (Bayes-Posterior and TS-Posterior), and the GP prediction of TRIO for the next task (Bayes-GP and TS-GP). For the first task, Bayes-GP and TS-GP are replaced by the initial prior given to the algorithm. For the AntGoal sequences, we report only the x-coordinate of the goal position.

track complex temporal patterns.

Figure 1 reports two sequences for AntGoal and one for HalfCheetahVel. As we can see, in all cases, TS-TRIO and Bayes-TRIO successfully track the changes occurring in the latent space. In HalfCheetahVel, our algorithms outperform state-of-the-art baselines. In this scenario, TS-TRIO achieves the best results. We conjecture that this happens due to its simpler task-conditioned policy model, which potentially leads to an easier training process that ends up in a slightly better solution. Interestingly, differently to what reported in [Zintgraf *et al.*, 2019], we also found RL2 to perform better than VariBAD. This might be due to the fact that we changed the reward signal, introducing stronger non-linearities. Indeed, VariBAD, which uses a reward decoder to train its inference network, might have problems in reconstructing this new function, leading to a marginally worse solution. Finally, MAML (Oracle) suffers from the same limitation as in the Minigolf domain.

In AntGoal, both our algorithms reach the highest performance. It is worth noting that, in line with the Minigolf domain, when the non-stationarity presents high discontinuities (as in sequence B), TRIO suffers a perfomance drop which is resolved in only a handful of episodes. MAML (Oracle) achieves competitive performance with VariBAD; however, we recall that MAML's results are shown post-adaptation, meaning that it has already explored the current task multiple times. Finally, being the problem more complex, RL2, compared to VariBAD, faces more troubles in the optimization procedure, obtaining a worse behavior.

Finally, it has to be highlighted that, in both problems, our algorithms are able to exploit the temporal patterns present in the non-stationarity affecting the latent variables. *Anticipating* the task evolution before it occurs leads to faster adaptation and higher performance.

6 Conclusions

We presented TRIO, a novel meta-learning framework to solve non-stationary RL problems using a combination of multi-task learning, curve fitting, and variational inference. Our experimental results show that TRIO outperforms state-of-the-art baselines in terms of achieved rewards during sequences of tasks faced at meta-test time, despite having no information on these sequences at training time. Tracking the temporal patterns that govern the evolution of the latent variables makes TRIO able to optimize for future tasks and leads to highly-competitive results, thus establishing a strong meta-learning baseline for non-stationary settings.

Our work opens up interesting directions for future work. For example, we could try to remove the need of task descriptors at training time, e.g., by building and tracking a reward-driven latent structure [Kamienny *et al.*, 2020] or a representation to reconstruct future rewards [Zintgraf *et al.*, 2019]. The main challenge would be to build priors over this learned latent space to be used for training the inference module.

References

[Al-Shedivat *et al.*, 2018] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *International Conference on Learning Representations*, 2018.

[Blei *et al.*, 2017] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112, 2017.

[Chandak *et al.*, 2020] Yash Chandak, Georgios Theocharous, Shiv Shankar, Martha White, Sridhar Mahadevan, and Philip Thomas. Optimizing for the future in non-stationary MDPs. In *International Conference on Machine Learning*, 2020.

[Clavera *et al.*, 2019] 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. *International Conference on Learning Representations*, 2019.

[Doshi-Velez and Konidaris, 2013] Finale Doshi-Velez and George Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. *IJCAI*, 2013.

[Duan *et al.*, 2016] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[Finn *et al.*, 2017] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

[Hochreiter *et al.*, 2001] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*. Springer, 2001.

[Hospedales *et al.*, 2020] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[Humplik *et al.*, 2019] Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

[Kamienny *et al.*, 2020] Pierre-Alexandre Kamienny, Matteo Pirotta, Alessandro Lazaric, Thibault Lavril, Nicolas Usunier, and Ludovic Denoyer. Learning adaptive exploration strategies in dynamic environments through informed policy regularization. *arXiv preprint arXiv:2005.02934*, 2020.

[Kaushik *et al.*, 2020] Rituraj Kaushik, Timothée Anne, and Jean-Baptiste Mouret. Fast online adaptation in robotics through meta-learning embeddings of simulated priors. *arXiv preprint arXiv:2003.04663*, 2020.

[Lan *et al.*, 2019] Lin Lan, Zhenguo Li, Xiaohong Guan, and Pinghui Wang. Meta reinforcement learning with task embedding and shared policy. *IJCAI*, 2019.

[Nagabandi *et al.*, 2018] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl. *International Conference on Learning Representations*, 2018.

[Padakandla, 2020] Sindhu Padakandla. A survey of reinforcement learning algorithms for dynamically varying environments. *arXiv preprint arXiv:2005.10619*, 2020.

[Puterman, 1994] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1994.

[Rakelly *et al.*, 2019] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, 2019.

[Rasmussen, 2003] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[Rothfuss *et al.*, 2019] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. *International Conference on Learning Representations*, 2019.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Thompson, 1933] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25, 1933.

[Tirinzoni *et al.*, 2019] Andrea Tirinzoni, Mattia Salvini, and Marcello Restelli. Transfer of samples in policy search via multiple importance sampling. In *International Conference on Machine Learning*, 2019.

[Wang *et al.*, 2016] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[Xie *et al.*, 2020] Annie Xie, James Harrison, and Chelsea Finn. Deep reinforcement learning amidst lifelong nonstationarity. *arXiv preprint arXiv:2006.10701*, 2020.

[Zintgraf *et al.*, 2019] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. *International Conference on Learning Representations*, 2019.