24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Model-Free Non-Stationarity Detection and Adaptation in Reinforcement Learning

Giuseppe Canonaco and **Marcello Restelli** and **Manuel Roveri**[1]

Abstract. In most Reinforcement Learning (RL) studies, the considered task is assumed to be stationary, i.e., it does not change its behavior or its characteristics over time, as this allows to generate all the convergence properties of RL techniques. Unfortunately, this assumption does not hold in real-world scenarios where systems and environments typically evolve over time. For instance, in robotic applications, sensor or actuator faults would induce a sudden change in the RL settings, while in financial applications the evolution of the market can cause a more gradual variation over time. In this paper, we present an adaptive RL algorithm able to detect changes in the environment or in the reward function and react to these changes by adapting to the new conditions of the task. At first, we develop a figure of merit onto which a hypothesis test can be applied to detect changes between two different learning iterations. Then, we extended this test to sequentially operate over time by means of the CUmulative SUM (CUSUM) approach. Finally, the proposed change-detection mechanism is combined (following an adaptive-active approach) with a well known RL algorithm to make it able to deal with non-stationary tasks. We tested the proposed algorithm on two well-known continuous-control tasks to check its effectiveness in terms of non-stationarity detection and adaptation over a vanilla RL algorithm.

1 Introduction

Continuously learning and adapting to the changes we face in everyday life is one of the huge differences between humans and machines. For example, a non-stationary behavior could be induced in a robotic system by a fault affecting sensors/electronics/actuators, by aging effects/thermal drift in the sensors or by seasonality/periodicity effects in the environment in which the robotic system operates. Unfortunately, non-stationarity is one of the most complex issues to deal with when attempting to solve a Reinforcement Learning (RL) problem, but, it is also one of the characteristics that, once properly addressed, will make an RL agent a step closer to a human being. Therefore, tackling and solving general non-stationary problems correctly would allow RL to reach an important milestone in the journey towards general artificial intelligence.

In recent years, a range of solutions designed to deal with non-stationarity in RL have been proposed. An extension of the Markov Decision Process (MDP) model [20], called Hidden Mode MDP (HMMDP), where non-stationarity is modeled through a Hidden Markov Chain is proposed in [6]. The HMMDP is learned through a variant of the Baum-Welch algorithm. In such a scenario, once a model of the environment, including the non-stationary dynamics, is learned, traditional model-based RL techniques could be considered. Unfortunately, the number of working modes or, equivalently, the number of changes in the environment must be known a-priori. This assumption rarely holds in the real world. In order to overcome the assumption of having a fixed and a-priori known number of environment changes, a solution based on the estimation of both the reward function and the environment transition-function was proposed in [7]. The main drawback of this solution lies in the way it performs the change detection, which is not theoretically-grounded (i.e., a heuristic is proposed) and based on several problem-dependent parameters. The previously mentioned solution was extended by using a CUmulative SUM (CUSUM) [2] sequential statistical test to perform the change detection in the environment [12]. Unfortunately, this solution still needs to estimate both the reward function and the state transition function of the various regimes it encounters. Moreover, this solution is meant to operate on finite MDPs. In order to deal with non-stationarity in RL, tracking is proposed in [25]. This technique is based on customizing the policy to the current situation via an adaptive learning rate. In other words, this solution can meta-learn the step-size to adequately adapt to the scenario it is facing. This technique does not detect the environment change, but simply mitigates the potentially catastrophic effect of non-stationarity on the algorithm performance.

A different line of research has been recently pursued in [11, 17]. These solutions are based on a passive adaptation to the changing environment through a sliding window. However, a fixed number of changes is assumed in [11], whereas an upper-bound on the amount of variation the reward function (or the state-transition function) can undergo is assumed to be known in [17], which is still not assumed by our approach. Moreover, both solutions [11, 17] deal with finite-state MDPs.

Finally, the solution proposed in this paper shows some affinities with fast-adaptation algorithms based on meta-learning [1, 16]. Anyhow, these algorithms assume to have access to the distribution over the tasks to face (through which they construct a meta-model able to nearly immediately adapt to new tasks coming from this distribution). Differently, we assume our learner has to deal with a non-stationary environment without any knowledge about the distribution over the possible tasks to face.

In this paper, we introduce a theoretically-grounded change-detection mechanism to detect non-stationarities related to performance degradation during the learning process of any RL algorithm. This tool, called Non-Stationarity Detector for Reinforcement Learning (NSD-RL), is able to detect changes in both the reward function and in the state transition function of the task the RL algorithm is learning, as long as the non-stationarity implies a variation

[1] Politecnico di Milano, Italy, emails: {giuseppe.canonaco, marcello.restelli, manuel.roveri}@polimi.it

in the agent performance[2]. More specifically, NSD-RL is hierarchically composed of two main modules. The lower module is a statistical hypothesis test coupled with an Importance Sampling (IS) [13] strategy, where the IS strategy will be used to transform two independent batches of data to allow the change detection. The upper module sequentially analyses the outcome of the low-level module by means of a CUSUM approach to detect non-stationarities in the RL problem. The proposed NSD-RL allows to trigger, following the active-adaptive approach introduced in [8], the reaction to the detected change and the adaptation of the RL algorithm by resetting the first and second moments of the Adam [14] optimizer. The effectiveness of the proposed NSD-RL is tested in two well-known continuous control RL tasks under the effect of different kinds of non-stationarities.

This paper is organized as follows. In Section 2, we describe the preliminaries and formulate the RL problem in non-stationary environments. In Section 3, we introduce the policy selection to support non-stationarity detection, while Section 4 details the optimization of the Reny divergence for policy selection. The change-detection and adaptation mechanism for RL is introduced in Section 5. Experimental results are given in Section 6 and conclusions are drawn in Section 7.

2 Preliminaries and Problem Formulation

In this section we initially introduce the RL framework; we then provide the theoretical background of the IS technique and we conclude by providing a formulation of the problem of RL in non-stationary environments.

2.1 Reinforcement Learning Background

A discrete-time continuous MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \rho\}$ can be used to model a Reinforcement Learning task [24]. $\mathcal{S}$ and $\mathcal{A}$ represent the state space and the action space, respectively, either continuous or discrete. $\mathcal{P}$ represents the Markovian transition function, where $\mathcal{P}(s'|s,a)$ is the transition density from state s to state s' given that the action a is executed. r with $r(s,a) \in [-R, R]$ represents the expected reward for the pair (s,a). $\gamma \in [0,1)$ and ρ are the discount factor and the initial state distribution, respectively.

The agent's behavior is modeled through a policy π, where $\pi(\cdot|s)$ describes a probability density function over the action space $\mathcal{A}$ given the state being currently visited. We are considering episodic MDPs with effective horizon H, hence a trajectory τ can be a finite sequence of states and actions $(s_0, a_0, s_1, a_1, \ldots, s_{H-1}, a_{H-1})$, where $s_0 \sim \rho$. Following a given policy, we can sample a trajectory from the environment. We denote by $p(\tau|\pi)$ the density distribution induced by policy π on the set $\mathcal{T}$ of all the possible trajectories defined as:

$$p(\tau|\pi) = \rho(s_0)\pi(a_0|s_0) \prod_{t=1}^{H} \mathcal{P}(s_t|s_{t-1}, a_{t-1})\pi(a_t|s_t).$$

$\mathcal{R}(\tau) = \sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)$ is the total discounted reward associated with trajectory τ. All policies can be ranked based on their expected total discounted reward: $J(\pi) = \mathbb{E}_{\tau \sim p(\cdot|\pi)}[\mathcal{R}(\tau)]$. Solving an RL task modeled through an MDP $\mathcal{M}$ means finding $\pi^* \in \arg\max_\pi \{J(\pi)\}$.

The most interesting application of NSD-RL refers to the *continual learning setting*. In this setting, Policy Gradient (PG) [26] techniques offer general and flexible solutions to RL problems and, for this reason, we will rely on this family of algorithms to present the proposed solution.

Policy gradient methods focus on searching for the best-performing policy over a set of parametrized policies $\Pi_{\boldsymbol{\theta}} = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \mathbb{R}^d\}$, with $\pi_{\boldsymbol{\theta}}$ differentiable w.r.t. $\boldsymbol{\theta}$. For brevity, the performance of a parametric policy will be denoted by $J(\boldsymbol{\theta})$ or equivalently by $J_{\boldsymbol{\theta}}$. Furthermore, the probability of a trajectory τ will be denoted by $p(\tau|\boldsymbol{\theta})$ or equivalently by $p_{\boldsymbol{\theta}}(\tau)$ (on some occasions, $p_{\boldsymbol{\theta}}(\tau)$ will be replaced by $p_{\boldsymbol{\theta}}$ omitting the dependence on τ for the sake of readability). Gradient ascent is used to find a locally optimal policy. The policy gradient is defined as [26, 18]:

$$\nabla J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau \sim p(\cdot|\boldsymbol{\theta})} \left[\nabla \log p_{\boldsymbol{\theta}}(\tau) \mathcal{R}(\tau)\right]. \tag{1}$$

At each iteration $i > 0$, a batch $\mathcal{D}_i^N = \{\tau_j\}_{j=0}^N$ of $N > 0$ trajectories is collected using policy $\pi_{\boldsymbol{\theta}_i}$. The policy is then updated as $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha \widehat{\nabla}_N J(\boldsymbol{\theta}_i)$, where α is a step size and $\widehat{\nabla}_N J(\boldsymbol{\theta})$ is an estimate of Eq. (1) on $\mathcal{D}_i^N$, i.e.,

$$\widehat{\nabla}_N J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{j=1}^{N} g(\tau_j|\boldsymbol{\theta}), \quad \tau_j \in \mathcal{D}_i^N, \tag{2}$$

where $g(\tau_j|\boldsymbol{\theta})$ is an estimate of $\nabla \log p_{\boldsymbol{\theta}}(\tau_j) \mathcal{R}(\tau_j)$. Depending on how we define the policy gradient estimator, we obtain different RL algorithms.

2.2 Importance Sampling

The idea behind the proposed NSD-RL is to evaluate the performance of a fixed policy using a pair of estimators fed with data coming from two different iterations of the RL algorithm. Therefore, we need a mechanism to take into account the fact that the data were sampled with different policies.

Let us assume that we want to find $\mathbb{E}_{x \sim P}[f(x)] = \int_{\mathcal{D}} f(x)p(x)dx$ where p is a probability density function on $\mathcal{D} \subseteq \mathbb{R}^d$ with $p(x) = 0, \ \forall x \notin \mathcal{D}$, then:

$$\begin{aligned} \int_{\mathcal{D}} f(x)p(x)dx = \int_{\mathcal{D}} \frac{f(x)p(x)}{q(x)} q(x)dx = \\ = \mathbb{E}_{x \sim Q}\left[\frac{f(x)p(x)}{q(x)}\right], \end{aligned} \tag{3}$$

where q is a positive probability density function on $\mathbb{R}^d$ such that $supp(q) \supset supp(p)$ and $\mathbb{E}_{x \sim Q}[\cdot]$ denotes expectation for $x \sim Q$. The IS technique introduces a multiplicative correction coefficient that compensates the fact that we are sampling from Q instead of sampling directly from P.

Importance sampling allows off-policy evaluation in RL [28, 27]. In the off-policy evaluation settings two policies, called behavioral π^B and target π^T, are involved. In this context, we aim at estimating the performance of the target policy π^T on samples collected using the policy π^B. We use IS to correct the fact that we sampled the trajectories using π^B and obtain an unbiased estimate of $J(\pi^T)$:

$$J(\pi^T) = \mathop{\mathbb{E}}_{\tau \sim p(\cdot|\pi^T)} [\mathcal{R}(\tau)] = \mathop{\mathbb{E}}_{\tau \sim p(\cdot|\pi^B)} \left[\omega_{T/B}(\tau)\mathcal{R}(\tau)\right], \tag{4}$$

where $\omega_{T/B}(\tau) = \frac{p(\tau|\pi^T)}{p(\tau|\pi^B)} = \prod_{t=0}^{H} \omega_{T/B}(s_t, a_t)$ and $\omega_{T/B}(s_t, a_t) = \frac{\pi^T(a_t|s_t)}{\pi^B(a_t|s_t)}$. In the context of our NSD-RL, we will

[2] We are not interested in non-stationarities not affecting the agent's performance since they are not so crucial in a real world scenario.

use the *per-decision* version of the above IS estimator that exploits the fact that a given reward should not be weighted according to the future of that trajectory, but only w.r.t. the likelihood of the trajectory up to that point [19].

2.3 Non-Stationarity in Reinforcement Learning: Problem Formulation

We can model non-stationarity in any RL task through a change in the Markovian state transition function $\mathcal{P}$ or in the reward function r. In the most general scenario, $\mathcal{P}$ and r may be both affected by non-stationarity (possibly at the same time). Therefore, there exists an iteration i^* of the RL algorithm after which the trajectories sampled from the environment will be, partially or totally, associated to a new task. All the trajectories in i^* are associated with the new task if the transition to this new task takes place at the beginning of the sampling procedure performed at step i^* of the RL algorithm itself, otherwise, just a subset of them will be associated to the new task. Since NSD-RL is not influenced by the latter situation we have just described, we will formulate non-stationarity in RL by assuming that the change in either $\mathcal{P}$ or r (or both) occurs at the beginning of the sampling process of a given optimization step of the algorithm. In other words, we can formalize non-stationarity in RL tasks as follows: for any $i < i^*$ we have $\mathcal{M}_1 = \{\mathcal{S}, \mathcal{A}, \mathcal{P}_1, r_1, \gamma, \rho\}$, whereas for $i \geq i^*$ we have $\mathcal{M}_2 = \{\mathcal{S}, \mathcal{A}, \mathcal{P}_2, r_2, \gamma, \rho\}$, where $\mathcal{P}_1 \neq \mathcal{P}_2 \vee r_1 \neq r_2$. We are assuming that ρ is not affected by the non-stationarity.

The goal of the proposed NSD-RL is to promptly detect changes in $\mathcal{M}$ without introducing false positive or negative detections. Such a change-detection represents also a crucial information to support the next adaptation and learning phase of the RL algorithm.

3 Policy Selection to support Non-Stationarity Detection

As mentioned in Section 2.3, if the task we are trying to solve is non-stationary, then it may happen that, between steps i and $i-1$, $\mathcal{P}$ or r (or both) may change. The first question we have to address is how to detect a change that can occur at any time during the execution of our RL algorithm? Answering this question is crucial to support an effective reaction and adaptation of the policy in a non-stationary environment. In order to reach our goal, we first need to be able to detect a change between two arbitrary fixed steps of the RL algorithm. Therefore, given data collected through policy $\pi_{\boldsymbol{\theta}_i}$ at step i and data collected through policy $\pi_{\boldsymbol{\theta}_{i-k}}$ at step $i-k$, we need to test $\mathcal{H}_0$*: there is no change in* $\mathcal{M}$ *between* i *and* $i-k$ against $\mathcal{H}_1$*: there is a change in* $\mathcal{M}$ *between* i *and* $i-k$. Before resorting to a statistical test, we need to spot the figure of merit on which to apply the test. This figure of merit is meant to operate on two independent datasets available in the two different steps of the RL algorithm. Therefore, by using the IS technique described in Section 2.2, we can write:

$$\bar{J}(\pi_\mu) = \mathop{\mathbb{E}}_{\tau\sim p(\cdot|\pi_\mu)}\left[\bar{\mathcal{R}}(\tau)\right] = \mathop{\mathbb{E}}_{\tau\sim p(\cdot|\pi_{\boldsymbol{\theta}_i})}\left[\omega_{p_\mu/p_{\boldsymbol{\theta}_i}}(\tau)\bar{\mathcal{R}}(\tau)\right] \tag{5}$$

$$\bar{J}(\pi_\mu) = \mathop{\mathbb{E}}_{\tau\sim p(\cdot|\pi_\mu)}\left[\bar{\mathcal{R}}(\tau)\right] = \mathop{\mathbb{E}}_{\tau\sim p(\cdot|\pi_{\boldsymbol{\theta}_{i-k}})}\left[\omega_{p_\mu/p_{\boldsymbol{\theta}_{i-k}}}(\tau)\bar{\mathcal{R}}(\tau)\right], \tag{6}$$

where $\bar{\mathcal{R}}(\tau) = \sum_{t=0}^{H-1} r(s_t, a_t)$ is the expected total undiscounted reward. If there are no changes in $\mathcal{M}$ between iteration i and $i-k$, the two expected values are equal, otherwise they are different[3]. Since we cannot exactly compute the expected values in Eq. (5) and (6), we resort to the associated estimators:

Table 1. Estimated type I error of the bootstrap test [10, chap. 16] under $\mathcal{H}_0$ w.r.t. different choices of π_μ. The two sampling policies, $\pi_{\boldsymbol{\theta}_{i-k}}$ and $\pi_{\boldsymbol{\theta}_i}$, are $\mathcal{N}$(10,13) and $\mathcal{N}$(-1,4), respectively. First row is associated with policy π_μ chosen optimizing Equation (9).

π_μ	Mean Type I error	Std. Dev.	Obj. Fun. (9)
$\mathcal{N}$(-0.3487, 4.846)	0.0462	0.0206	3.95
$\mathcal{N}$(4, 4.5)	0.0578	0.0206	9.21
$\mathcal{N}$(8, 2)	0.109	0.0315	31.96
$\mathcal{N}$(8, 5)	0.1687	0.0333	128237.5

$$\hat{\bar{J}}_{\mu/\boldsymbol{\theta}} = \frac{1}{N}\sum_{j=0}^{N} \omega_{\mu/\boldsymbol{\theta}}(\tau_j)\bar{\mathcal{R}}(\tau_j), \tag{7}$$

where N is the number of trajectories sampled using $\pi_{\boldsymbol{\theta}}$. Then, our goal is to properly choose the policy π_μ in order to have an effective hypothesis test able to detect changes in $\mathcal{M}$ between iteration i and $i-k$. From [15] we know that:

$$\mathbb{V}ar\left[\hat{\bar{J}}_{\mu/\boldsymbol{\theta}}\right] \leq \frac{1}{N}||r||_\infty^2 d_2(p_\mu||p_{\boldsymbol{\theta}}), \tag{8}$$

where $d_2(p_\mu||p_{\boldsymbol{\theta}})$ is the exponetiated Renyi divergence [21] of the distribution induced by policy π_μ from the distribution induced by policy $\pi_{\boldsymbol{\theta}}$. Therefore, choosing the policy π_μ such that:

$$\pi_\mu^* \in \arg\min_{\pi_\mu} d_2(p_\mu||p_{\boldsymbol{\theta}_i}) + d_2(p_\mu||p_{\boldsymbol{\theta}_{i-k}}) \tag{9}$$

will allow us to minimize the upper bound on the variance of the estimators of Eq. (5) and (6), hence increasing the power of our hypothesis test.

The optimization task in Eq. (9) aims at picking a policy π_μ such that we do not have unbounded weights when using IS, which means that the estimators of Eq. (5) and (6) converge smoothly in the number of samples. Note that having unboundend weights might induce the estimators of Eq. (5) and (6) to abruptly change as we increase the number of samples [22, chap. 3]. In this case, we cannot rely on the smooth convergence properties of the estimators. In other words, we will likely end up with an estimate very far from the real value, which in turn severely affects the ability of the hypothesis test to keep the type I error under control (see Table 1). We should also observe that if the two sampling policies, $\pi_{\boldsymbol{\theta}_i}$ and $\pi_{\boldsymbol{\theta}_{i-k}}$, are very far from each other (in terms of Renyi divergence) there will be no policy π_μ able to induce good behavior in the IS procedure (an example of this behavior is experimentally given in Table 2). Observe that, in order to produce both Table 1 and 2, we have used the *”Guess a Number”* task. This is a single state task where the reward function is $r(a) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(a-\mu)^2}{2\sigma}}$ for a fixed μ and σ which define the task. The agent will get higher rewards executing actions which are as close as possible to μ. Therefore, the optimal policy consists in always playing $a = \mu$. As an example, in the context of the above described experiments, we have $r(a) = \frac{1}{10\sqrt{2\pi}}e^{-\frac{(a+4)^2}{20}}$. This setting, while being rather simple, already confirms the complexity of selecting an evaluation policy without side effects on the type I error.

We emphasize that the estimators for Eq. (5) and (6) share the expected value but they are characterized by different and unknown probability distributions. In fact, we do not have any a-priori information about the family of probability distributions the estimators for Eq. (5) and (6) belong to, e.g., we cannot assume they are Gaussian.

[3] Notice that if the transition between the tasks happens in the middle of the sampling procedure at step i, the two expected values will still be different.

Table 2. Estimated type I error of the bootstrap test [10, chap. 16] under $\mathcal{H}_0$ increasing the distance of policy $\pi_{\boldsymbol{\theta}_{i-k}}$ from $\pi_{\boldsymbol{\theta}_i}$ which instead remains fixed to $\mathcal{N}$(-1,4). The mean policy π_μ is always chosen optimizing Equation (9).

$\pi_{\boldsymbol{\theta}_i}$	Mean Type I error	Std. Dev.	Obj. Fun. (9)
$\mathcal{N}$(1, 13)	0.0478	0.0231	3.12
$\mathcal{N}$(10, 13)	0.0462	0.0206	3.95
$\mathcal{N}$(35, 13)	0.0334	0.017	47.23
$\mathcal{N}$(50, 13)	0.0087	0.009	586.86
$\mathcal{N}$(100, 13)	0	0	7931313070349.74

Hence, in order to support our analysis, we need a statistical hypothesis test able to detect variations in the expected value without making any assumption about the underlying probability distributions. We emphasize that not satisfying the assumptions of the statistical hypothesis test would induce critical issues in controlling the type I error. In our specific case, such a problem would severely affect the sequential analysis characterizing NDS-RL that will be described in Section 5. For these reasons, we resort to a bootstrap hypothesis test proposed by [10, chap. 16] which, being a test for detecting variations in the mean of two arbitrary distributions, perfectly fits into the context we are working with. Hence, in the scenario of RL in non-stationary environments, we define the hypothesis test as a function

$$Te(\mathcal{D}^N_{\mu/\boldsymbol{\theta}_i}, \mathcal{D}^N_{\mu/\boldsymbol{\theta}_{i-k}}, \alpha) = \begin{cases} +1, & \text{if we reject } \mathcal{H}_0 \\ -1, & \text{otherwise} \end{cases} \tag{10}$$

where $\mathcal{D}^N_{\mu/\boldsymbol{\theta}_i}$ and $\mathcal{D}^N_{\mu/\boldsymbol{\theta}_{i-k}}$ are the datasets sampled at iterations i and $i-k$, respectively, after applying IS, and $\mathcal{H}_0$ represents the null hypothesis, i.e., the two expected values are equal. If the output of this function is +1, then a change is detected between iterations i and $i-k$ of the RL algorithm given a confidence level α otherwise no change occured. We would like to stress the fact that NSD-RL is able to detect only changes affecting the performance of the agent, here represented by the expected total undiscounted reward. However, changes not affecting the performance are less relevant to be detected in a real world scenario.

4 Renyi Divergence Optimization

In the previous section, we defined an objective function aiming at selecting the policy π_μ to be used in the hypothesis test defined in Eq. (10). In this section, we propose a way to optimize such an objective function. Notice that computing the Renyi divergence between two distributions over trajectories is, of course, intractable also given the transition density of the task we are currently solving. For this reason the following estimator was introduced in [15]:

$$\hat{d}_2(p_\mu || p_{\boldsymbol{\theta}}) = \frac{1}{N}\sum_{j=1}^{N}\prod_{t=0}^{H-1} d_2(\pi_\mu(\cdot|s_{\tau_j,t})||\pi_{\boldsymbol{\theta}}(\cdot|s_{\tau_j,t})). \tag{11}$$

If we plug Eq. (11) into the optimization problem stated in Eq. (9) we get:

$$\begin{aligned}\pi^*_\mu &\in \arg\min_{\pi_\mu} \hat{d}_2(p_\mu||p_{\boldsymbol{\theta}_i}) + \hat{d}_2(p_\mu||p_{\boldsymbol{\theta}_{i-k}}) = \\ &= \arg\min_{\pi_\mu} \frac{1}{N}\sum_{j=1}^{N}\Bigg(\prod_{t=0}^{H-1} d_2(\pi_\mu(\cdot|s_{\tau_j,t})||\pi_{\boldsymbol{\theta}_{i-k}}(\cdot|s_{\tau_j,t})) + \\ &\qquad \prod_{t=0}^{H-1} d_2(\pi_\mu(\cdot|s_{\tau_j,t})||\pi_{\boldsymbol{\theta}_{i-k}}(\cdot|s_{\tau_j,t}))\Bigg).\end{aligned} \tag{12}$$

Now we can solve a separate optimization problem for each trajectory since we have independent variables for each $s_{\tau_j,t}$:

$$\begin{aligned}\arg\min_{\pi_\mu} \Bigg(&\prod_{t=0}^{H-1} d_2(\pi_\mu(\cdot|s_{\tau,t})||\pi_{\boldsymbol{\theta}_{i-k}}(\cdot|s_{\tau,t})) + \\ &\prod_{t=0}^{H-1} d_2(\pi_\mu(\cdot|s_{\tau,t})||\pi_{\boldsymbol{\theta}_{i-k}}(\cdot|s_{\tau,t}))\Bigg), \\ &\forall \tau \in \mathcal{D}^N_i \cup \mathcal{D}^N_{i-k}.\end{aligned} \tag{13}$$

We reformulated the problem in the following way to allow independent optimization over all the states $s_{\tau,t}$:

$$\begin{aligned}\arg\min_{\pi_\mu} \Bigg(&d_2(\pi_\mu(\cdot|s_{\tau,t})||\pi_{\boldsymbol{\theta}_{i-k}}(\cdot|s_{\tau,t})) + \\ &d_2(\pi_\mu(\cdot|s_{\tau,t})||\pi_{\boldsymbol{\theta}_{i-k}}(\cdot|s_{\tau,t}))\Bigg), \\ &\forall \tau \in \mathcal{D}^N_i \cup \mathcal{D}^N_{i-k} \forall t = 0 \cdots H-1.\end{aligned} \tag{14}$$

We emphasize that in RL with PG we have two popular choices for the set of smoothly parametrized policies $\Pi_{\boldsymbol{\theta}}$: Gibbs policies and Gaussian policies. The first one is generally used when we have a finite number of actions executable on the environment by the agent, while the second one is employed whenever the set of executable actions is infinite. Since we are dealing with finite-horizon MDPs, each trajectory τ, sampled from the environment, will have a maximum number H of visited states. For what concern Gaussian (or Gibbs) policies, we have a parametrized Gaussian (or Gibbs) distribution for which the Renyi divergence can be computed analytically in each state. Since we have a per-state analytical form of the Renyi divergence, we will find the π_μ only for those states stored in $\mathcal{D}^N_i$ and $\mathcal{D}^N_{i-k}$. It is worth noting that we do not have the parametrized Gaussians (or Gibbs) generated by policy $\pi_{\boldsymbol{\theta}_{i-k}}$ in the context of the states in $\mathcal{D}^N_i$ and the same holds for $\pi_{\boldsymbol{\theta}_i}$ in the states in $\mathcal{D}^N_{i-k}$. However, the missing parametrizations can be computed straightforwardly without any interactions with the environment. Notice that π_μ is only used for non-stationarity detection, and not to perform actions on the environment. In Section 4.1, we will focus on Gaussian policies, whereas, in Section 4.2, we provide an optimal solution to the problem described in Eq. (14) in the context of Gibbs policies.

4.1 Optimization for Gaussian Policies

Once we fix a state s, we have an analytical expression for the two terms of the objective function in Eq. (14) [5]:

$$\begin{aligned}&d_2(\mathcal{N}(\mu,\Sigma)||\mathcal{N}(\mu_{\boldsymbol{\theta}_i},\Sigma_{\boldsymbol{\theta}_i})) = \\ &\frac{|\Sigma_{\boldsymbol{\theta}_i}|}{\sqrt{|\Sigma||\Sigma^*_{\boldsymbol{\theta}_i}|}} e^{(\mu-\mu_{\boldsymbol{\theta}_i})^\intercal \Sigma^{*-1}_{\boldsymbol{\theta}_i}(\mu-\mu_{\boldsymbol{\theta}_i})} \\ &d_2(\mathcal{N}(\mu,\Sigma)||\mathcal{N}(\mu_{\boldsymbol{\theta}_{i-k}},\Sigma_{\boldsymbol{\theta}_{i-k}})) = \\ &\frac{|\Sigma_{\boldsymbol{\theta}_{i-k}}|}{\sqrt{|\Sigma||\Sigma^*_{\boldsymbol{\theta}_{i-k}}|}} e^{(\mu-\mu_{\boldsymbol{\theta}_{i-k}})^\intercal \Sigma^{*-1}_{\boldsymbol{\theta}_{i-k}}(\mu-\mu_{\boldsymbol{\theta}_{i-k}})},\end{aligned}$$

where $|\cdot|$ denotes the determinant of a matrix, $\Sigma^*_{\boldsymbol{\theta}_i} = 2\Sigma_{\boldsymbol{\theta}_i} - \Sigma$ and $\Sigma^*_{\boldsymbol{\theta}_{i-k}} = 2\Sigma_{\boldsymbol{\theta}_{i-k}} - \Sigma$ assuming that both $\Sigma^*_{\boldsymbol{\theta}_i}$ and $\Sigma^*_{\boldsymbol{\theta}_{i-k}}$ are positive-definite. In order to match this assumption, we restrict the optimization procedure over all the possible solutions having a diagonal Σ. In this way, we only need to satisfy the constraints on the diagonal elements of Σ:

$$\sigma^{jj} < \min(\sqrt{2}\sigma^{jj}_{\boldsymbol{\theta}_i}, \sqrt{2}\sigma^{jj}_{\boldsymbol{\theta}_{i-k}}) \wedge \sigma^{jj} > 0 \qquad \forall j = 1 \dots dim(\mathcal{A}), \tag{15}$$

where $dim(\mathcal{A})$ denotes the action space dimension[4]. Now we can resort to any optimization procedure present in the literature provided that it supports bounds on the objective function domain.

4.2 Optimization for Gibbs Policies

Following the rationale of the previous section, we will show how to optimize the Renyi divergence with Gibbs policies on a per state basis. Given a state s, we have a parametrized categorical distribution assigning a certain probability to each action available in s. Therefore, denoting with $P_{\boldsymbol{\theta}_i} = (p_{\boldsymbol{\theta}_i,1}, \dots, p_{\boldsymbol{\theta}_i,n})$ and $P_{\boldsymbol{\theta}_{i-k}} = (p_{\boldsymbol{\theta}_{i-k},1}, \dots, p_{\boldsymbol{\theta}_{i-k},n})$ the categorical distributions parametrized by $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{i-k}$, respectively, in state s we can write Equation (14) as follows:

$$\operatorname*{arg\,min}_{P_\mu} \quad \sum_{h=0}^{n} \frac{p^2_{\mu,h}}{p_{\boldsymbol{\theta}_i,h}} + \sum_{h=0}^{n} \frac{p^2_{\mu,h}}{p_{\boldsymbol{\theta}_{i-k},h}} \tag{16}$$

$$subject\ to: \quad \sum_{h=0}^{n} p_{\mu,h} = 1, \tag{17}$$

$$p_{\mu,h} \geq 0\ \forall\ h. \tag{18}$$

If we do not take into account the constraint (18), we have a relaxed problem which can be solved by using Lagrangian multipliers:

$$L = \sum_{h=0}^{n} \frac{p^2_{\mu,h}}{p_{\boldsymbol{\theta}_i,h}} + \sum_{h=0}^{n} \frac{p^2_{\mu,h}}{p_{\boldsymbol{\theta}_{i-k},h}} + \lambda \left(\sum_{h=0}^{n} p_{\mu,h} - 1 \right), \tag{19}$$

taking derivatives we have:

$$\frac{\partial L}{\partial p_{\mu,h}} = 2 \left(\frac{1}{p_{\boldsymbol{\theta}_i,h}} + \frac{1}{p_{\boldsymbol{\theta}_{i-k},h}} \right) p_{\mu,h} + \lambda = 0\ \forall\ h \tag{20}$$

$$\frac{\partial L}{\partial \lambda} = \sum_{h=0}^{n} p_{\mu,h} - 1 = 0, \tag{21}$$

and now solving:

$$p_{\mu,h} = \frac{1}{\left(\frac{p_{\boldsymbol{\theta}_i,h} + p_{\boldsymbol{\theta}_{i-k},h}}{p_{\boldsymbol{\theta}_i,h} p_{\boldsymbol{\theta}_{i-k},h}} \right) C} \tag{22}$$

$$\lambda = -\frac{2}{C}, \tag{23}$$

where

$$C = \sum_{h=0}^{n} \frac{p_{\boldsymbol{\theta}_i,h} p_{\boldsymbol{\theta}_{i-k},h}}{p_{\boldsymbol{\theta}_i,h} + p_{\boldsymbol{\theta}_{i-k},h}}. \tag{24}$$

Since the objective function in Eq. (16) is convex and the solution we have just found also satisfies the constraint in Eq. (18), then the solution we found is also optimal and unique for the non-relaxed problem.

5 Change-Detection and Adaptation Mechanism for Reinforcement Learning

The hypothesis test aiming at detecting a non-stationarity between step i and $i-k$ defined in Eq. (10) is here extended to operate sequentially by introducing a sequential change-detection mechanism based on the well-known and theoretically-grounded CUSUM [2, 23]. More specifically, the proposed CUSUM-based NSD-RL mechanism operates as follows. Let $\bar{i}$ be the reference iteration initially set to a point where the agent has reached convergence. $\bar{i}$ represents the iteration at which we activate the NSD-RL change detection mechanism. The sequential analysis of $\mathcal{M}$ is performed over windows of length $2k$ (being $k \in \mathbb{N}^+$) and relies on the computation of a figure of merit m_i able to take into account the outcome of the hypothesis test $Te(\mathcal{D}^N_{\mu/\boldsymbol{\theta}_i}, \mathcal{D}^N_{\mu/\boldsymbol{\theta}_{i-k}}, \alpha)$ applied sequentially to detect a non-stationarity, i.e.,

$$m_i = max \left(0, m_{i-1} + Te(\mathcal{D}^N_{\mu/\boldsymbol{\theta}_i}, \mathcal{D}^N_{\mu/\boldsymbol{\theta}_{i-k}}, \alpha) \right), \tag{25}$$

with $i = \bar{i} + k, \dots, \bar{i} + 2k - 1$ and being $m_{\bar{i}+k-1} = 0$. This allows us to take into account the first window ranging from $\bar{i}$ to $\bar{i} + 2k - 1$. Once we have analysied the first window, the algorithm switches to the next window and m_{i-1} withhold the last value of the figure of merit on the previous window. Notice that this window-based approach allows us to keep independent all the different tests we perform in the sequential analysis.

A change is detected at the i-th iteration when,

$$m_i \geq K, \tag{26}$$

being $K \in \mathbb{N}^+$ a change-detection threshold, which is set at design time. The choice of K is crucial to trade-off false positive detections (i.e., detections of changes before i^*) and false negative detections (i.e., changes are not detected by the change-detection mechanism). In our analysis, such a choice is supported by the theoretical analysis of the mean time to a false positive detection, i.e., the Average Run Length (ARL$_0$), provided in [23] stating that

$$ARL_0(\alpha) = \underline{u}(I - P_\alpha)^{-1}\underline{1} \tag{27}$$

where I is the $(K+1) \times (K+1)$ identity matrix, P_α is the $(K+1) \times (K+1)$ matrix defined as follows

$$P_\alpha = \begin{bmatrix} 1-\alpha & \alpha & 0 & \dots & 0 \\ 1-\alpha & 0 & \alpha & \dots & 0 \\ \vdots & \vdots & & \vdots & \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix},$$

$\underline{1}$ is the $(K+1)$-dimensional vector of ones, and $\underline{u}$ is the $(K+1)$-dimensional vector defined as $\underline{u} = [1, 0, \dots, 0]$, being α the confidence level of the hypothesis test stated in Eq. (10). In our scenario ARL$_0(\alpha)$ refers to the mean number of executions of the hypothesis test before the NSD-RL raises a false positive detection. Setting the expected ARL$_0$ (that is application-specific) allows to identify the corresponding value of the threshold K.

The proposed NSD-RL change-detection mechanism operating on a generic PG algorithm is shown in Algorithm 1. More specifically, Lines 4 and 21 refer to the PG implementation. Lines 5 - 20 implement the computation of the figure of merit m_i as described above. More precisely, in Line 6 we store the data in order to perform the hypothesis tests, in Line 9 we compute π_μ solving the optimization problem stated in Eq. (14), in Line 10 the hypothesis

[4] Notice that in a monodimensional action space the restricted optimization problem is equivalent to the not restricted one.

Algorithm 1 PG-NSD-RL

```
1: Input: change-detection threshold $K$, confidence level $\alpha$, step size $\eta$, policy initialization $\boldsymbol{\theta}_0$, batch size N, number of epochs $I$, reference epoch $\bar{i}$, distance between epochs $k$
2: $B = \{\phi\}$
3: for $i = 0$ to $I - 1$ do
4:   sample $N$ trajectories $D_i^N = \{\tau_j\}_{j=1}^N$ from $p(\cdot|\boldsymbol{\theta}_i)$
5:   if $i >= \bar{i}$ and $i < \bar{i} + k$ then
6:     $B = B \cup (\mathcal{D}_i^N, \boldsymbol{\theta}_i)$
7:   end if
8:   if $(i >= \bar{i} + k)$ then
9:     compute $\pi_\mu$ according to Eq. (14) and apply IS on both $\mathcal{D}_i^N$ and $\mathcal{D}_{i-k}^N$
10:    $m_i = max\left(0, m_{i-1} + Te(\mathcal{D}^N_{\mu/\boldsymbol{\theta}_i}, \mathcal{D}^N_{\mu/\boldsymbol{\theta}_{i-k}}, \alpha)\right)$, see Eq. (25)
11:    if $i - \bar{i} == 2k - 1$ then
12:      $\bar{i} + = 2k$
13:      $B = \{\phi\}$
14:    end if
15:    if $m_i > K$ then
16:      Change detected
17:      Reset configuration for the next detection
18:      Resetting Adam
19:    end if
20:  end if
21:  $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i + \eta\widehat{\nabla}_N J(\boldsymbol{\theta})$
22: end for
23: return $\pi_{\boldsymbol{\theta}^*}$
```

test $Te(\mathcal{D}^N_{\mu/\boldsymbol{\theta}_i}, \mathcal{D}^N_{\mu/\boldsymbol{\theta}_{i-k}}, \alpha)$ is evaluated and m_i is computed as described in Eq. (25), in Line 12 we move to the next window and in Line 13 we reset the buffer. Finally, in Line 15, the value of m_i is tested w.r.t. K to detect a change. Once the change has been detected, in Line 17 we reset all the configurations for the next change-detection phase and in Line 18 we react to the change to compensate as fast as possible the loss in performance. In the context of this work, we adopted a straightforward adaptation technique that consists in resetting the Adam [14] optimizer by erasing its history related to the first and second moments, allowing it to forget what it currently knows about the behavior of the gradients in the previous task, which in turn implies a greater reactivity of the optimizer in the new task.

6 Experiments

In this section, we evaluate the improvement in performance of NSD-RL over G(PO)MDP [3], which is a traditional non-adaptive RL algorithm. More precisely, G(PO)MDP is a refinement of REINFORCE [29] exploiting the fact that the current reward does not depend on future actions. In other words the gradient estimator of G(PO)MDP performs a proper credit assignment, which may imply a variance reduction on the gradient estimate itself. G(PO)MDP is coupled with the average discounted reward baseline to further reduce the gradient estimator variance. In order to make a fair comparison the learning algorithm used in PG-NSD-RL is also G(PO)MDP. Both PG-NSD-RL and G(PO)MDP use Adam [14] as optimizer endowing them with an adaptive learning rate. The considered RL tasks are *Pendulum-v0* [4] and *Mountain Car* [9] that are widely used RL tasks in the related literature. *Pendulum-v0* consists of a classical pendulum swing-up problem. The goal of the agent is to keep the pendulum in an upright position via the application of forces to the pendulum. The observation space is a 3-dimensional vector composed of $\cos\theta$, $\sin\theta$, and the pole velocity $\dot{\theta}$. The monodimensional action is the force applied to the pendulum by the agent. The reward $r(s,a) = -(\theta^2 + 0.1\dot{\theta}^2 + 0.001a^2)$. *Mountain Car*, instead, consists in escaping a valley via the application of limited tangetial forces. Due to this limitation, the car has to alternately drive up along the two slopes of the valley in order to gain sufficient momentum to overcome gravity. The observation space is made of a 2-dimensional vector composed of the horizontal position, x, and the horizontal velocity, $\dot{x}$, of the car. The reward function is $r(s,a) = -1 + height$, where $height$ is the car's vertical offset. The G(PO)MDP parametrization is shown in Table 3. Notice that this configuration is shared by the baseline (i.e., vanilla G(PO)MDP) and by our proposed algorithm (PG-NSD-RL). Moreover, while our algorithm is at regime, Adam's learning rate is fixed to 10^{-5} in order to prevent the policies from different iteration to be too far from each others, whereas, when our algorithm detects a change, Adam is reset to the initial conditions stated in Table 3.

Table 3. G(PO)MDP experimental configuration

Parameter	*Pendulum-v0*	*Mountain Car*
Neural Network hidden weights	(32,32)	(32,32)
Neural Network activation function	tanh	tanh
Batch size N	100	100
Task horizon	200	500
Discount factor γ	0.99	0.99
Adam β_1	0.9	0.9
Adam β_2	0.999	0.999
Adam α	0.005	0.005

The experiments have been organized as follows. Each run comprises at most $I = 500$ learning iterations for *Pendulum-v0* and $I = 300$ for *Mountain Car*. The learning process begins at iteration $i = 0$. The algorithm NSD-RL is activated at iteration $\bar{i}$. A change in the state transition function $\mathcal{P}$ is introduced at iteration $i^* = \bar{i} + 50$. Let $\hat{i}$ be the iteration at which NSD-RL detects a change, we define a false positive detection when $\hat{i} < i^*$ and a false negative detection when $\hat{i} > I - 1$. A correct detection is considered when $i^* \leq \hat{i} \leq I - 1$ and, in this case, we compute the detection delay as $\hat{i} - i^*$.

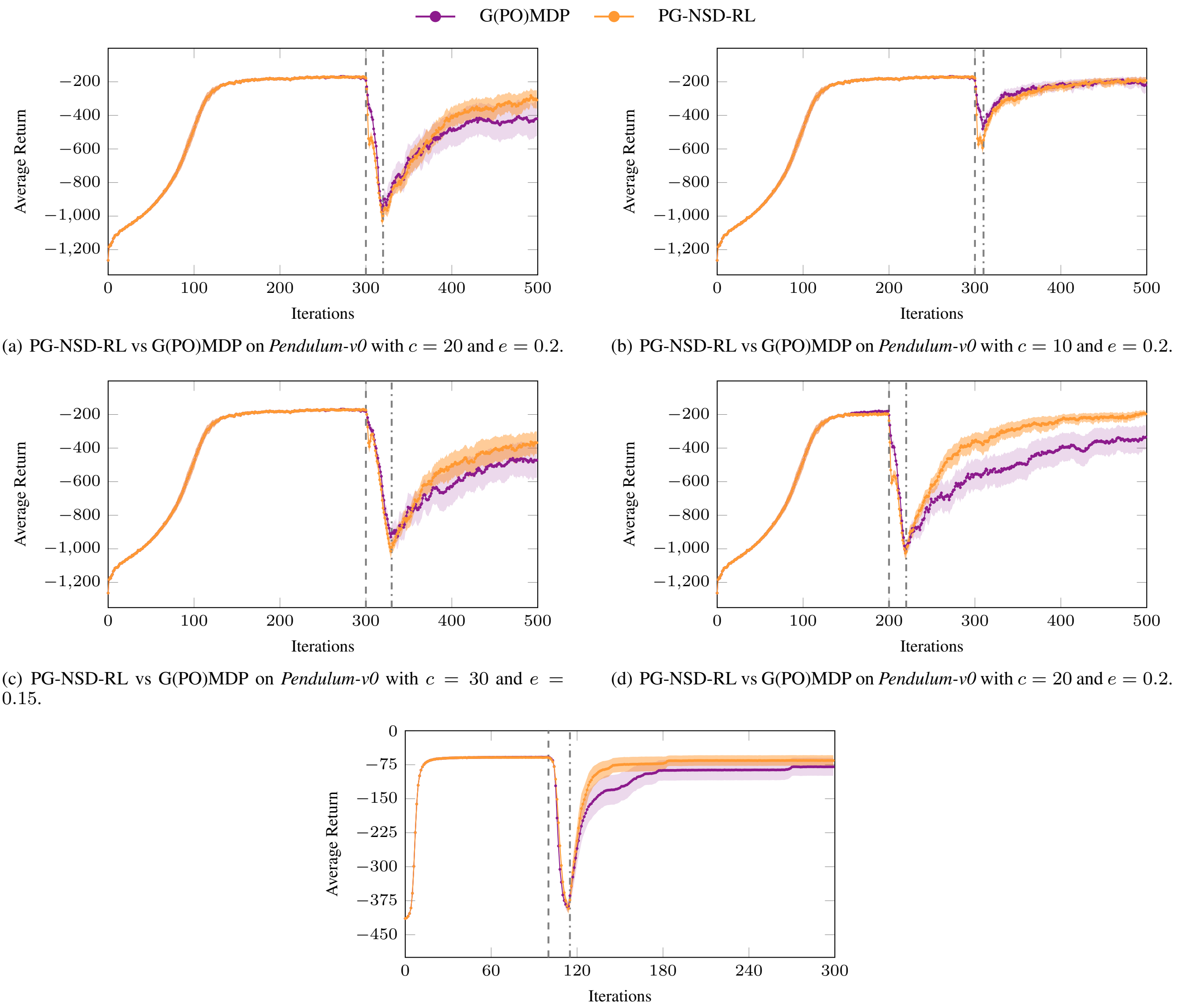
Let $s = [\cos\theta, \sin\theta, \dot{\theta}]$ be the vector representing the state in the *pendulum-v0* task, to model non-stationarity we have considered an additive clamped-ramp perturbation affecting $\dot{\theta}$ after i^* defined as follows:

$$s = \begin{cases} [\cos\theta, \sin\theta, \dot{\theta}] & \text{if } i < i^* \\ [\cos\theta, \sin\theta, \dot{\theta} + \nu] & \text{otherwise }, \end{cases} \tag{28}$$

where

$$\nu = \begin{cases} e(i - i^*), \text{ if } i - i^* <= c \\ e \cdot c \text{ otherwise,} \end{cases} \tag{29}$$

being e the speed of the anomaly and c the duration of its transient component. In particular, we considered three different configurations of ν, i.e., $(e = 0.2, c = 20)$, $(e = 0.2, c = 10)$ and $(e = 0.15, c = 30)$. Moreover, we have considered two different onset points of the anomaly $i^* = 200$ and $i^* = 300$. In the context of *Mountain Car*, the perturbation is still an additive clamped-ramp similarly to what defined above, but it affects $\dot{x}$. For this task we have considered one configuration of ν, i.e., $(e = 0.09, c = 15)$. Furthermore, for this experiment the onset point of the anomaly is $i^* = 100$.



(a) PG-NSD-RL vs G(PO)MDP on *Pendulum-v0* with $c = 20$ and $e = 0.2$.

(b) PG-NSD-RL vs G(PO)MDP on *Pendulum-v0* with $c = 10$ and $e = 0.2$.

(c) PG-NSD-RL vs G(PO)MDP on *Pendulum-v0* with $c = 30$ and $e = 0.15$.

(d) PG-NSD-RL vs G(PO)MDP on *Pendulum-v0* with $c = 20$ and $e = 0.2$.

(e) PG-NSD-RL vs G(PO)MDP on *Mountain Car* with $c = 15$ and $e = 0.09$.

Figure 1. Comparison of on-line performance over the iterations of the optimization algorithm, with 90% t-student confidence intervals. The first vertical line (dashed) highlight the injection point, whereas the second vertical line (dotted-dashed) highlight the end of the transient part of the anomaly.

The parameter K of NSD-RL has been set to 3, while $\alpha = 0.05$ and $k = 10$ in all the experiments. For all the configurations, we considered 50 runs with different seeds and average results are shown in Figure 1.

Two main comments arise. In Figures 1(a), 1(c), 1(d) and 1(e), we can see how PG-NSD-RL shows a better behavior in terms of average undiscounted return after i^*. This is due to a prompt detection combined with the adaptation guaranteeing a higher plasticity of PG-NSD-RL. Interestingly, in Figure 1(b) we show that PG-NSD-RL and G(PO)MDP have similar performances. This is due to the the fact that the anomaly does not induce a relevant change on the environment. Furthermore, in Figures 1(a) and 1(d), we can see how changing the onset point of the anomaly has a huge impact on performance. Indeed, in the first case the neural network parametrizing the policy reaches a worse configuration of the weights w.r.t. the second case due to a greater number of regime updates, which in some sense could be thought as overfitting the environmental noise in the task.

In Table 4, we show some indicators assessing the quality of the detection phase of the proposed change detection algorithm. As we can see, FPRs are equal for all the configurations. This is reasonable since false positive detections do not depend on the type of change. Moreover, we can see how the detection delay increases from $(e = 0.2, c = 20, i^* = 300)$ to $(e = 0.15, c = 30, i^* = 300)$. This is due to the fact that e is smaller in the second configuaration (i.e., $e = 0.15, c = 30$) inducing a more gradual drift which is, of course, more subtle to be detected. Notice that in Table 4 we have not reported the results relative to the configuration $(e = 0.2, c = 10, i^* = 300)$ since they are equivalent to those of $(e = 0.2, c = 20, i^* = 300)$.

This is reasonable since we have only decreased the duration of the transient part and the maximum detection delay in the configuration $(e = 0.2, c = 20, i^* = 300)$ is 7. Finally, as expected, the two configurations, $(e = 0.2, c = 20, i^* = 200)$ and $(e = 0.2, c = 20, i^* = 300)$, have very similar average and standard deviation for the detection delay. In the last row of Table 4 we show how our NSD-RL algorithm is able to properly detect non-stationarities in another task.

Table 4. NSD-RL performance in terms of False Positive Rate (FPR), False Negative Rate (FNR) and Detection Delay (DD) in different scenarios.

ν	i^*	Task	FPR	FNR	DD	DD std.
$e = 0.2, c = 20$	300	*Pendulum-v0*	0	0	4.26	0.955
$e = 0.15, c = 30$	300	*Pendulum-v0*	0	0	4.76	1.141
$e = 0.2, c = 20$	200	*Pendulum-v0*	0	0	3.74	0.743
$e = 0.09, c = 15$	100	*Mountain Car*	0	0	4.66	0.839

7 Conclusions

The aim of this paper was to introduce a change-detection mechanism to detect changes in a RL problem and integrate it into a RL algorithm to deal with non-stationary tasks. The proposed change-detection mechanism relies on the joint use of a statistical hypothesis test (aiming at comparing the expected value of the reward at two different iterations) and a CUSUM-based sequential mechanism to detect changes in the MDP. Whereas the adaptation phase relies on resetting the history associated to the moments of the Adam optimizer in order to increase the plasticity of the algorithm. The proposed solution is theoretically-grounded and has been successfully tested in two well-known RL tasks showing performance improvements over G(PO)MDP.

The next steps of this work will encompass the design of a transfer learning algorithm to be included into the adaptation phase of the current proposed solution. Furthermore, we will extend the detection mechanism endowing it with a diagnostic part able to identify and characterize the type, temporal evolution and magnitude of the change. This latter diagnostic tool will represent a valuable information for the adaptation phase.

REFERENCES

[1] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel, ‘Continuous adaptation via meta-learning in nonstationary and competitive environments’, *arXiv preprint arXiv:1710.03641*, (2017).

[2] Michèle Basseville, Igor V Nikiforov, et al., *Detection of abrupt changes: theory and application*, volume 104, Prentice Hall Englewood Cliffs, 1993.

[3] Jonathan Baxter and Peter L Bartlett, ‘Infinite-horizon policy-gradient estimation’, *Journal of Artificial Intelligence Research*, **15**, 319–350, (2001).

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, ‘Openai gym’, *arXiv preprint arXiv:1606.01540*, (2016).

[5] Jacob Burbea, ‘The convexity with respect to gaussian distributions of divergences of order α’, *Utilitas Mathematica*, **26**, 171–192, (1984).

[6] Samuel PM Choi, Dit-Yan Yeung, and Nevin Lianwen Zhang, ‘An environment model for nonstationary reinforcement learning’, in *Advances in neural information processing systems*, pp. 987–993, (2000).

[7] Bruno C Da Silva, Eduardo W Basso, Ana LC Bazzan, and Paulo M Engel, ‘Dealing with non-stationary environments using context detection’, in *Proceedings of the 23rd international conference on Machine learning*, pp. 217–224. ACM, (2006).

[8] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar, ‘Learning in nonstationary environments: A survey’, *IEEE Computational Intelligence Magazine*, **10**(4), 12–25, (2015).

[9] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel, ‘Benchmarking deep reinforcement learning for continuous control’, in *International Conference on Machine Learning*, pp. 1329–1338, (2016).

[10] Bradley Efron and Robert J Tibshirani, ‘An introduction to the bootstrap, volume 57 of’, *Monographs on Statistics and applied probability*, 17, (1993).

[11] Pratik Gajane, Ronald Ortner, and Peter Auer, ‘A sliding-window algorithm for markov decision processes with arbitrarily changing rewards and transitions’, *arXiv preprint arXiv:1805.10066*, (2018).

[12] Emmanuel Hadoux, Aurélie Beynier, and Paul Weng, ‘Sequential decision-making under non-stationary environments via sequential change-point detection’, in *Learning over Multiple Contexts (LMCE)*, (2014).

[13] Timothy Classen Hesterberg, *Advances in importance sampling*, Ph.D. dissertation, Stanford University, 1988.

[14] Diederik P Kingma and Jimmy Ba, ‘Adam: A method for stochastic optimization’, *arXiv preprint arXiv:1412.6980*, (2014).

[15] Alberto Maria Metelli, Matteo Papini, Francesco Faccio, and Marcello Restelli, ‘Policy optimization via importance sampling’, in *Advances in Neural Information Processing Systems*, pp. 5442–5454, (2018).

[16] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn, ‘Learning to adapt in dynamic, real-world environments through meta-reinforcement learning’, *arXiv preprint arXiv:1803.11347*, (2018).

[17] Ronald Ortner, Pratik Gajane, and Peter Auer, ‘Variational regret bounds for reinforcement learning’, in *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence*, (2019).

[18] Jan Peters and Stefan Schaal, ‘Reinforcement learning of motor skills with policy gradients’, *Neural networks*, **21**(4), 682–697, (2008).

[19] Doina Precup, Richard S. Sutton, and Satinder P. Singh, ‘Eligibility traces for off-policy policy evaluation’, in *ICML*, (2000).

[20] Martin L Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 2014.

[21] Alfréd Rényi, ‘On measures of information and entropy’, in *Proceedings of the 4th Berkeley symposium on mathematics, statistics and probability*, volume 1, (1961).

[22] Christian Robert and George Casella, *Monte Carlo statistical methods*, Springer Science & Business Media, 2013.

[23] Manuel Roveri, ‘Learning discrete-time markov chains under concept drift’, *IEEE transactions on neural networks and learning systems*, (2019).

[24] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, volume 1, MIT press Cambridge, 1998.

[25] Richard S Sutton, Anna Koop, and David Silver, ‘On the role of tracking in stationary environments’, in *Proceedings of the 24th international conference on Machine learning*, pp. 871–878. ACM, (2007).

[26] 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*, pp. 1057–1063, (2000).

[27] Philip Thomas and Emma Brunskill, ‘Data-efficient off-policy policy evaluation for reinforcement learning’, in *International Conference on Machine Learning*, pp. 2139–2148, (2016).

[28] Philip S Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh, ‘High-confidence off-policy evaluation’, in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, (2015).

[29] Ronald J Williams, ‘Simple statistical gradient-following algorithms for connectionist reinforcement learning’, *Machine learning*, **8**(3-4), 229–256, (1992).