Bayesian Online Non-Stationary Detection for Robust Reinforcement Learning

Alexander Shmakov **Pankaj Rajak** **Yuhao Feng** **Wojciech Kowalinski** **Fei Wang**
Amazon
Seattle, WA, USA
{ashmakov, rajakpan, yuhaof, kowalin, fiwan }@amazon.com

Abstract

Reinforcement Learning (RL) has achieved state-of-the-art performance in stationary environments with effective simulators. However, lifelong and open-world RL applications, such as robotics, stock trading, and recommendation systems, change over time in adversarial ways. Non-stationary environments pose challenges for RL agents due to constant distribution shifts from the training data, leading to deteriorating performance. We propose using a robust Bayesian online detector, which tracks agent performance to detect non-stationarities in the environment. Additionally, we propose a new metric called hindsight approximate reward (HAR) that solely relies on state and action information to detect adversarial changes in the environment, making it well-suited for real-world settings with missing or delayed feedback. We demonstrate that the proposed Bayesian detector, combined with HAR or expected reward as a metric, can detect a range of non-stationary changes in dynamic control tasks more effectively compared to baseline non-stationary tests.

1 Introduction and Literature Review

Reinforcement Learning (RL) presents methods for training optimal agents in static Markov Decision Processes (MDPs). RL solutions are proven to be successful in many applications, including tree-search board games [1], console video games [2, 3, 4], robotics [5, 6], manufacturing and various engineering fields. Traditional RL approaches assume that the training environment matches the eventual deployment environment. However, many open-ended and real-world applications of RL, such as advertise ranking, content recommendation, bidding, stock trading, and physical robotics, have either dynamically evolving or adversarial environments [7, 8, 9, 10, 11]. Agents which do not account for these environments' constant shifts can suffer unbounded regret over time, inciting a need to detect and adapt to such differences in their policy.

Non-stationary RL (Non-Sta RL) aims to tackle this problem by training agents in static MDPs while remaining adaptable to changes in the underlying MDP dynamics. Here, a general design consists of detection of environment change followed by retraining agent in the new environment. However, most work done in Non-Sta RL makes further assumptions about the environment such as tabular MDPs with only discrete actions [12], environment's switch from a fixed number of MDPs [13], environment drifting slowly over time [14] or Bandits [15]. These Non-Sta solutions focus on measuring regret [16], solving for exact optimal policy in a sliding window fashion regardless of environment change [12], or modeling reward and environment dynamics to predict Non-Sta points [17]. Such approach can be challenging to apply in practical applications with highly complex state-action space and delayed or potentially inaccessible reward.

In this work, we propose a robust RL agent that implicitly contains a Non-Sta detector to identify changes in the environment's dynamics in the open-world setting or reward structures in real-

Accepted at the Intrinsically Motivated Open-ended Learning workshop at NeurIPS 2024.

Algorithm 1 Non-Stationarity Measure-Detector Framework

Require: N_{train}, N_{adapt}, MEASURE(s, a, r, s), DETECTION(x)
  π_θ - Agent trained on N_{train} environment trajectories.
  DETECTION - Detection algorithm fine-tuned on N_{adapt} environment trajectories.
  $M \leftarrow [\,]$
  $NS \leftarrow False$
  while $\neg NS$ **do**
    $Traj \leftarrow [\,]$
    $s \leftarrow s_0$
    while $\neg terminal$ **do**
      $a \leftarrow \pi(s)$
      $s', r, terminal \leftarrow$ STEP(s, a)
      $Traj \leftarrow Traj + (s, a, r, s')$
      $s \leftarrow s'$
    end while
    $M \leftarrow M +$ MEASURE$(Traj)$
    $NS \leftarrow$ DETECTION(M)
  end while

time. Specifically, we focus on black-box non-stationarities in general function-approximation RL, assuming no additional information about the environment beyond the standard observation and reward signals. We focus our attention on jump non-stationarity, where the change in the environment happens suddenly and at an unknown point in time, in contrast to drift non-stationarities which occur slowly. We present a Non-Sta detector using Bayesian online change-point detection [18, 19] for regret-free, black-box non-stationarity detection. The detector tracks several metrics from environment, probing both situations where reward is available during evaluation and when the reward is available during training-time only. We also propose a new metric called hindsight approximate reward to track intrinsic non-stationary changes and is useful when explicit reward signal from environment is unavailable. Finally, we present results evaluating different general signals and detection techniques on a classical control environments with different sized non-stationarities.

2 Methods

We present a general framework for detecting non-stationarities in black-box environments. Since individual transitions are very noisy and often ill-informative, we operate on the level of trajectories. This allows us to more generally determine if a change has occurred between a trajectory boundary. We assume transitions occur between trajectories and do not occur within a trajectory. This approach requires defining two components:

- **Measure**: A (potentially multi-variate) real-valued summary of the agent's performance, $m(t) \in \mathbb{R}^n$, which can be extracted from a trajectory of transitions (s, a, r, s'). We also examine the case where we only have access to (s, a, s') transitions.
- **Detection**: An algorithm which accepts a time-series of measure values $m(t)$ and determines if a change has occurred in the previous time-step.

A full description of the non-stationary RL evaluation with detection is presented in Algorithm 1.

2.1 Measures

We define several measures and compare them across several tasks. Measures should be rapidly computable from a trajectory and noise-free to ensure a reliable detection without false positives. Each measure calculation is given by a list of transitions $[(s_t, a_t, r_t, s'_t)]$ from a contiguous trajectory. Additionally, any method should be independent of the base RL agent chosen for the policy. As such we do not assume the existence of a Q function, probabilistic policy π, or any other aspects of the base policy. Following these limitations, we present several black-box measures using only this trajectory data.

- **Reward (R)** $\mathbb{E}[r_t]$: A simple measure of average reward across a trajectory. This signal will operate as a baseline, and the simplest black-box measure across environments.
- **Approximate Reward (AR)** $\mathbb{E}[\hat{r_t}(s_t, a_t)]$: Train a reward model along-side the RL agent, which estimates the reward function of the MDP. This will be imperfect but will still allow us to apply our techniques in environments with no inference-time reward available.
- **Hindsight Approximate Reward (HAR)** $\mathbb{E}[\hat{r_t}(s_t, s'_t)]$: An alternative approximate reward function which estimates the reward from two consecutive states instead of a state-action. The main difference in this measure is it can only be calculated in hindsight after an action has been completed and is suitable when environment feedback is delayed or missing.
- **Reward Error (RE)** $\mathbb{E}[r_t - \hat{r_t}(s_t, a_t)]$: To remove natural stationary drift in the environment due to agent action randomness, we propose to track the temporal difference error in the reward model through time. This also has a benefit of making the signal zero-centered by default.
- **Hindsight Reward Error (HRE)** $\mathbb{E}[r_t - \hat{r_t}(s_t, s'_t)]$: Equivalent to RE but with the hindsight reward model.

Normalization All measures are normalized using the same training trajectory data used for training the RL agent. We assume that the RL agent is trained during a period of stationarity, and normalize the measures to have zero mean and unit variance within this data.

Trajectory Aggregation We evaluate change-points on a trajectory level. Therefore, we must summarize the measure value across the entire trajectory before feeding it to the detectors. We elect to compute the measures on a step-by-step basis and then find the average value across an entire trajectory. This gives us an estimate of the expected value of each measure for a given policy.

2.2 Detectors

BOCPD We present using Bayesian online change-point detection (BOCPD) [18] to determine a non-stationary from a collection of metrics. BOCPD detects if a time-series has drifted out of distribution by tracking the posterior likelihood of all sub-sequences of the data following a given exponential family. By leveraging online Bayesian parameter updating, which is possible with exponential families, and tracking the posterior probabilities of all sequence lengths in an online fashion, BOCPD is able to process observations in $O(n)$ time, reducible to $O(1)$ time through the use of pruning [19]. A detailed overview of BOCPD is presented in Appendix A. We elect to use a multivariate Gaussian family to model our data and evaluate the following variants.

1. **($\mathbb{I}$-BOCPD)** A Gaussian likelihood with known unit variance and unknown mean following a unit Gaussian prior resulting in a Gaussian posterior.
2. **(Σ-BOCPD)** A Gaussian likelihood with unknown variance following an inverse Wishart prior and unknown mean following a unit Gaussian. The posterior for this likelihood is a multivariate Student-t distribution.
3. **($\mathcal{D}_m$-BOCPD)** A $\mathcal{D}_m$-Posterior variant of the Gaussian likelihood which uses an alternative Bayesian Update step to promote a more robust detection which avoids triggering on outliers. This results in a likelihood with no analytical form, but with a truncated normal posterior on the variance and normal posterior on the mean [19].

Baseline We also compare these methods with two baselines based on simple threshold detection.

- **Threshold**: A simple detector tracking if the normalized average measure ever passes a fixed threshold value: $\textsc{Threshold}(M) := |\mathbb{E}[M]| > \tau$. Effective and historically used for zero-expectation measures such as regret. The detector tuning phase adjusts the threshold based on a tuning dataset and observed values.
- **Difference**: A time-difference approach which measures the change between the current measure and the previous measure: $\textsc{Difference}(M_t) := |\mathbb{E}[M_t] - \mathbb{E}[M_{t-1}]| > \delta$. Useful for tracking sudden changes in performance metrics such as reward. The threshold is similarly adjust during tuning.

Mass Scale	0.5	0.75	0.8	0.9	0.95	1.0	1.05	1.1	1.5	2.0	5.0
R	1	**0**	**0**	4	14	∞	**6**	2	**0**	**0**	**0**
AR	**0**	**0**	**0**	2	∞	27	∞	12	-2	∞	-32
HAR	**0**	**0**	**0**	3	**2**	∞	∞	**1**	-22	**0**	**0**
RE	**0**	1	1	9	∞	30	∞	4	3	1	**0**
HRE	**0**	**0**	1	**1**	7	∞	11	3	**0**	**0**	**0**
RS	**0**	**0**	**0**	**1**	∞	36	∞	-44	45	∞	**0**

Table 1: Median Detection Latency for the $\mathbb{I}$-BOCPD detector across measures on `cartpole_swingup`. Negative Latency implies a false-positive early detection. A value of 0 implies a perfect detection after one episode. A mass scale of 1.0 implies no change actually occurred at the non-stationarity, and we expect this to be undetectable. The Lowest delay latency for each environment is **bolded**.

3 Results and Discussion

Experiment We evaluate our methods on the `cartpole_swingup` task from the `dm-control` [20] suite of classical control robotics problems. Episodes are 1000 steps long, with reward equal to the shifted angular distance from a fully vertical pole.

Changepoint Experiment Setup We introduce a non-stationarity by modifying the pole mass and therefore changing the dynamics of the environment. We sweep the pole mass from the original environment value of 1.0 to selected values in the range 0.1 and 10.0. Each non-stationarity only changes the mass one time for each experiment, and the new mass remains for the remainder of the episodes. We set the true change-point time at 100 episodes after starting evaluation, and present the difference between the detected change-point and the true change-point time. If a detector did not trigger after all 200 episodes, the latency is set as ∞.

Non-stationary Detectors We design a non-stationary detection experiment by collecting 150 episodes from the original environment and 100 more episodes from a modified mass environment. The first 50 episodes are used for tuning detector parameters, and we then evaluate each measure and detector on the sampled sequence of 100 + 100 original and modified episodes, determining when each configuration detects a change-point. Each experiment is repeated 11 times and the median detection time for each distinct mass is presented.

Detector Tuning We use the same set of base hyper-parameters for all methods and metrics. Most of our detectors assume a roughly unit Gaussian input distribution on the data. Large input values could throw off the detectors and cause false positives. To prevent this, detectors are tuned on the 50 initial episodes through a scaling technique. The tuning data is repeatedly scaled by a reduction factor until the detection method shows no false positives on the entire tuning period. We use a reduction factor of $\frac{1}{2}$.

RL Agent We use a standard Soft Actor-Critic (SAC) [6] agent as implemented by the TorchRL library [21]. SAC parameters are provided in the Appendix. The agent is trained for 3.5 Million environment steps in the original unit mass environment. The inference agent is deterministic, using the mean of the policy prediction as the action. We add the reward and hindsight reward networks to the SAC agent, and these are trained along-side the other components from replay buffer samples.

Measure Comparison We present a comparison of measures in Table 1. We notice that the reward signal is very effective, as expected. We also find that **HAR** is as effective as the true reward (**R**), while the **AR** model is much noisier and less effective. **HAR** does not depends on access to the reward function and estimates reward based only on state s_t and next state $s_t^{'}$ distribution change. Therefore, **HAR** will be a better metric in delayed or missing feedback settings. We also notice that when the reward is available, it is also better to use **HRE** likely because it removes the variance from the reward in the middle of the episode.

Mass Scale	0.5	0.75	0.8	0.9	0.95	1.0	1.05	1.1	1.5	2.0	5.0
$\mathbb{I}$-BOCPD	**1**	**0**	**0**	4	14	∞	6	2	**0**	**0**	**0**
Σ**-BOCPD**	2	**0**	**0**	3	16	∞	**0**	**1**	**0**	**0**	**0**
$\mathcal{D}_m$**-BOCPD**	**1**	**0**	1	**2**	**7**	∞	2	**1**	**0**	**0**	**0**
Threshold	**1**	-31	**0**	6	∞	∞	2	-36	-13	**0**	**0**
Difference	-12	∞	**0**	∞	∞	∞	-26	20	**0**	**0**	**0**

Table 2: Median Detection Latency for the reward **(R)** measure across tested detectors on `cartpole_swingup`. The remaining table definitions are identical to Table 1. The Lowest delay latency for each environment is **bolded**.

Detectors Comparison Detectors are compared on the baseline **R** measure in Table 2. We notice that all of the Bayesian detectors fared the best, with no early detection and low latencies. The robust $\mathcal{D}_m$-BOCPD seems to fare the best on this environment, but this may not be a general result. We found that the baseline methods suffer from the high variance between episodes, and even after tuning, the Threshold and Difference methods often had false positives. This happens because both Threshold and Difference use a point estimate to detect environment change with one single trajectory from the environment at time t, whereas $\mathcal{D}_m$-BOCPD uses all trajectories seen so far $(\leq t)$ to build a time-series model at multiple granularity to detect change at time t. To reduce this high variance issue with Threshold and Difference, we would need to sample multiple trajectories from the environment at time t but this inherently increases sample complexity and latency.

4 Conclusion and Future Work

Overview We present results on detecting discrete non-stationarities in environment dynamics at unknown time in the general reinforcement learning setting with access to only the reward or an approximation of the reward. We find that Bayesian change-point detection methods are effective in detecting RL non-stationarities from these (approximate) reward signals, and are promising on a practical application. We focus on the black box setting to ensure this method is widely applicable to a variety of RL tasks. We present experiments on one common environment and plan on applying to more fundamental non-stationary shifts in the future.

Use in Intrinsic Learning In an adversarial environment, which is commonly observed in open-world setting, RL agents can quickly find themselves in out-of-distribution states, which must be detected to remain safe, and the agents can then adapt accordingly. Also, discovering new phenomenon may present itself as non-stationary changes. In addition, we may use this detection to assist in adaption to new scenarios. Detection allows us to determine which episodes are from the new environments and adapt to these new situations in an intelligent manner. We leave this adaption step to future work, as the focus here is on robust detection of environment non-stationarities.

Limitations We note a limitation of these methods being the inability to capture a change in sub-optimal action rewards: where actions which the trained agent will not take suddenly receive very high reward. We believe that comparison to exploration agents, which will eventually execute all actions, is necessary to capture these types of changes. We leave this modification for future work.

References

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[2] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[3] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024.

[4] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018.

[5] Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations*, 2024.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[7] Weinan Zhang, Yifei Rong, Jun Wang, Tianchi Zhu, and Xiaofan Wang. Feedback control of real-time display advertising. *arXiv preprint arXiv:1603.01055*, 2016.

[8] Xun Yang, Yasong Li, Hao Wang, Di Wu, Qing Tan, Jian Xu, and Kun Gai. Bid optimization by multivariable control in display advertising. *arXiv preprint arXiv:1905.10928*, 2016.

[9] Yujian Ye, Dawei Qiu, Mingyang Sun, Dimitrios Papadaskalopoulos, and Goran Strbac. Deep reinforcement learning for strategic bidding in electricity markets. *IEEE Transactions on Smart Grid*, 2020.

[10] Sahin Lale, Kamyar Azizzadenesheli, Babak Hassibi, and Anima Anandkumar. Reinforcement learning with fast stabilization in linear dynamical systems. *arXiv preprint arXiv:2007.12291*, 2020.

[11] Tamas Jambor, Jun Wang, and Neal Lathia. Using control theory for stable and efficient recommender systems. *arXiv preprint arXiv:2007.12291*, 2012.

[12] Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Reinforcement learning for non-stationary markov decision processes: The blessing of (more) optimism. *arXiv preprint arXiv:2006.14389*, 2020.

[13] Sindhu Padakandla, Prabuchandran K. J, and Shalabh Bhatnagar. Reinforcement learning in non-stationary environments. *arXiv preprint arXiv:1905.03970*, 2020.

[14] Yash Chandak, Georgios Theocharous, Shiv Shankar, Martha White, Sridhar Mahadevan, and Philip S. Thomas. Optimizing for the future in non-stationary mdps. *arXiv preprint arXiv:2005.08158*, 2020.

[15] Qingyun Wu, Naveen Iyer, and Hongning Wang. Learning contextual bandits in a non-stationary environment. *arXiv preprint arXiv:2005.08158*, 2018.

[16] Chen-Yu Wei and Haipeng Luo. Non-stationary reinforcement learning without prior knowledge: An optimal black-box approach. *arXiv preprint arXiv:2102.05406*, 2021.

[17] Bruno C. da Silva, Eduardo W. Basso, A. Bazzan, and P. Engel. Dealing with non-stationary environments using context detection. *International Conference on Machine Learning preprint DOI 10.1145/1143844.1143872*, 2006.

[18] Ryan Prescott Adams and David J. C. MacKay. Bayesian online changepoint detection, 2007.

[19] Matias Altamirano, Briol François-Xavier, and Jeremias Knoblauch. Robust and scalable bayesian online changepoint detection. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[20] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

[21] Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. Torchrl: A data-driven decision-making library for pytorch, 2023.

Algorithm 2 Bayesian Online Changepoint Detection (BOCPD)

1. Define the initial conditions.
$$P(r_0 = 1) = 1, \mu_0 = \mathbf{0}, \Sigma_0 = \mathbb{I}$$
2. Observe a new datum x_t.
3. Compute the data posterior probability:
$$\pi_t = p(x_t) \text{ such that } x_t \sim \mathcal{N}(\mu_t, \Sigma_t)$$
4. Calculate the change in run-length probabilities. For each r_{t-1}:
$$P(r_t = r_{t-1} + 1, x_{1:t}) = P(r_{t-1}, x_{1:t-1})\pi_t(1 - H(r_{t-1}))$$
$$P(r_t = 0, x_{1:t}) = \sum_{r_{t-1}} P(r_{t-1}, x_{1:t-1})\pi_t H(r_{t-1})$$
5. Calculate the run-length posterior:
$$p(r_t) = \frac{p(r_t, x_{1:t})}{\sum_{r_t} p(r_t, x_{1:t})}$$
6. Update the data posterior parameters according to the chosen family's update rule.
7. If there exists a run-length $p(r_t \neq t) > p(r_t = t)$ for a specified number of updates (known as a lag), then we declare a changepoint.

Appendix

A Bayesian Online Change-point Detector

We provide a brief overview of the BOCPD algorithm used in this work. Complete descriptions of the methods may be found in the original papers [18, 19].

BOCPD requires selecting a base exponential family for describing the data $x(t)$. We use the multivariate Gaussian family with either known or unknown variance. The full prior distribution for the time-series is defined as the following system, where $\mathcal{W}$ is the Inverse-Wishart Distribution.

$$\begin{aligned} x &\sim \mathcal{N}(\mu, \Sigma) \\ \mu &\sim \mathcal{N}(\mathbf{0}, \mathbb{I}) \\ \Sigma &\sim \mathcal{W}(\Psi, \nu) \\ &\text{or} \\ \Sigma &= \mathbb{I} \end{aligned}$$

The posterior update rule for the prior distribution may now be defined. We start with the Known Variance Gaussian. Given an initial prior mean of $\mu_0 = \mathbf{0}$ and prior variance of $\Sigma_0 = \mathbb{I}$, after observing a new datapoint x_t, the parameters are updated as:

$$\begin{aligned} \Sigma_{t+1} &= \left(\Sigma_t^{-1} + \mathbb{I}\right)^{-1} \\ \mu_{t+1} &= \Sigma_{t+1}\left(\Sigma_t^{-1}\mu_t + \mathbb{I}x_t\right) \\ x_{t+1} &\sim \mathcal{N}(\mu_{t+1}, \Sigma_{t+1} + \mathbb{I}) \end{aligned} \tag{1}$$

Similar update rules may be derived for the multivariate Gaussian family with unknown likelihood [18] and the robust version of this update rule derived using $\mathcal{D}_m$-divergence in [19].

Once the data distribution has been defined, BOCPD introduces a *hazard function* for providing a prior on the run-length distribution of each sub-sequence fit. We choose a constant hazard function of $H(t) = \frac{1}{24}$ Hours.

Finally, BOCPD provides the full online Bayesian update for tracking the posterior distribution over run-lengths given the exponential family defined above [18]. The full psuedo-code for BOCPD is presented in Algorithm 2.