Decision making without prior knowledge in dynamic environments using Bootstrapped DQN

Anonymous Authors

Abstract

Learning how to adapt and make real-time informed decisions in unpredictable and dynamic environments to obtain maximal rewards is a non-trivial task. Reinforcement Learning (RL) agents interact with such an environment and learn optimal decision making policy through trial and error. This premise leads to considering multiple changes in the ways the environment interacts, without an agent having the perfect knowledge about it. In these scenarios, the development of a framework, which can detect such dynamic behaviour is crucial. In this paper, we put forward a holistic algorithm that can determine changes in environment's behavior and incrementally improvises on its policies. In particular, we introduce a Q-learning based framework to achieve efficient deep exploration of the action space, whereby multiple instances of Bootstrapped Deep Q-Network (DQN) are scheduled to facilitate a dynamic detection of changes in environment's behavior. At each time step, the proposed method takes actions according to the policy of the currently active agent for decision-making and is equipped with a change detection and adaptation policy, which does not depend on prior knowledge of the degree of uncertainty associated with the environment.

Introduction

In static environments, AI agents can follow rigid rules in order to execute certain tasks. However, in dynamic real-world environment, unexpected changes (i.e., novelties) can occur, and the AI agent is expected to detect and adapt to these changes in a timely manner. Recently, there has been significant interest in building self-supervised learning systems that can perform amidst the uncertainty of real-world tasks such as Chatbots (Pereira 2016) and self-driving cars (Haliem, Aggarwal, and Bhargava 2021). In an open-world, the agent interacts with the environment without having the complete knowledge about this environment, but learns through experience. This leads to a fundamental trade-off of exploration versus exploitation (Liu et al. 2020). The agent may improve its future rewards by brute force exploration of understood states and actions, but this may require sacrificing immediate rewards such as arriving at a decision. When it comes to exploitation, the search is limited to high probability blocks of actions. But exploitation needs specific criteria to select the right action space, which is difficult in a non-stationary environment. Despite the RL advances, detecting changes (novelties) and adapting to them in timely manner is a significant challenge, particularly in dynamic real-world mission critical systems. To learn efficiently an agent should explore only when there are valuable learning opportunities. In non-stationary environments, learning would be the most beneficial when a change (i.e., novelty) occurs in the environment. Therefore, in addition to extended deep exploration, the agent also exploits to detect the change.

Common dithering strategies, such as ϵ-greedy, approximate the value of an action by a numeric reward. Most of the time, the action with the highest reward is selected, but sometimes another action is chosen at random. In this paper, we adopt bootstrap with random initialization as it can produce reasonable uncertainty estimates for neural networks at low computational cost (Osband et al. 2016). Bootstrapped DQN leverages these uncertainty estimates for efficient (and deep) exploration, which in turn reduces learning times and improves performance across most games.

Bootstrapped DQN

For efficient exploration, the agent needs to quantify uncertainty in value estimates to be able to judge the potential benefit of exploratory actions. The bootstrap principle is to approximate a population distribution by a sample distribution (Efron and Tibshirani 1994). The network consists of a shared architecture with *K* bootstrapped "heads" (neurons or nodes) branching off independently. Each head is trained only on its bootstrapped sub-sample of the data and represents a single bootstrap sample $\psi(D)$. The shared network learns a joint feature representation across all the data, which can provide significant computational advantages at the cost of lower diversity between heads. Bootstrapped DQN explores in a manner similar to the provably-efficient algorithm Posterior Sampling for Reinforcement Learning (PSRL) (Osband, Russo, and Van Roy 2013), but it uses a bootstrapped neural network to approximate a posterior sample for the value. Unlike PSRL, bootstrapped DQN directly samples a value function and so does not require further planning steps. Therefore, bootstrapped DQN drives deep exploration.

Non-Stationary Detection and Adaptation

To be able to detect novelties in non-stationary environments, it is intuitive to minimize dynamic regret, which is the gap between the total reward of the optimal sequence of policies and that of the learner. There are several works in literature that adopt this approach such as: (Ortner, Gajane, and Auer 2020; Cheung, Simchi-Levi, and Zhu 2020; Domingues et al. 2021). However, the common issue with all these works is that they rely heavily on having prior knowledge on the degree of uncertainty of the world, such as how much or how many times the distribution changes, which is often unavailable in real-time.

In this work, we overcome this by adopting the black-box reduction approach proposed in (Wei and Luo 2021), applying it to model-free RL environments using Bootstrap DQN as our base algorithm. This approach is shown to achieve optimal dynamic regret without any prior knowledge on the degree of uncertainty. Here, the fundamental principle is to schedule multiple instances of Bootstrapped DQN with different intervals in a carefully-designed randomized scheme, which facilitates non-stationary detection with little overhead. Unlike the approaches in (Jun et al. 2017; Daniely, Gonen, and Shalev-Shwartz 2015), our approach does not try to learn the best instance; instead, it follows the action suggested by the instance with the current shortest scheduled interval, and only updates this instance after receiving feedback from the environment. The reason is the instances with shorter duration are responsible for detecting larger distribution changes. In addition, it ensures that the algorithms do not get blocked by the large interval instances and thus every scale of distribution change is detected in a timely manner.

Problem Definition:

We consider a reinforcement learning (RL) framework that covers a wide range of problems. Ahead of time, the learner is given a policy set Π, and the environment decides T reward functions $f_1, ..., f_T : \Pi \rightarrow [0, 1]$ unknown to the learner. Then, in each round $t = 1, ...T$, the learner chooses a policy $\pi_t \in \Pi$ and receives a reward R_t whose mean is $f_t(\pi_t)$. One way to measure non-stationarity is by measuring the distribution drift between rounds t and $t+1$ by observing how much the expected reward of any policy could change, that is, $max_{\pi\in\Pi}|f_t(\pi)-f_{t+1}(\pi)|$. However, to make our approach more general, we take a slightly more abstract way to define non-stationarity using dynamic regret. The dynamic regret of the learner is defined as D-REG = $\sum_{t=1}^{T}(f_t^* - R_t)$, where $f_t^* = max_{\pi\in\Pi} f_t(\pi)$ is the expected reward of the optimal policy for round t.

Proposed Approach:

MALG (Algorithm 2) is an algorithm that schedules and runs several instances of Bootstrap DQN in a multi-scale manner. MALG runs for an interval of length 2^n, which is called a block, for some integer n (unless it is terminated by non-stationary detection). During initialization, MALG uses Algorithm 1 to schedule multiple instances of Bootstrap DQN within the block such that: for every $m = n, n-1, ..., 0$, partition the block equally into 2^{n-m} sub-intervals of length 2^m, and for each of these sub-intervals, with probability $2^{-\frac{n-m}{2}}$, schedule an instance of Bootstrap DQN (otherwise skip this sub-interval). We use $alg.s$ and $alg.e$ to denote the start and end time of a specific instance.

In each time t, the unique instance covering this time step with the shortest length is considered as being active, while all others are inactive. This makes the time complexity of our algorithm linear so that it can perform reasonably well in real-time. MALG follows the decision of the active instance, and update it after receiving feedback from the environment. All inactive instances do not make any decisions or updates. We use $\tilde{g}_t$ to denote the scalar output by the active instance.

Algorithm 1: Scheduling Procedure

Input: n, I set of Bootstrapped DQN algorithm instances.
for $\tau = 0, 1, ...2^{n-1}$ **do**
  for $m = n, n-1, ..., 0$ **do**
    if τ *is a multiple of* 2^m **then**
      Schedule an algorithm (alg) using **Rule 1** with probability $2^{-(n-m)/2}$, such that it starts at $alg.s = \tau + 1$ and ends at $alg.e = \tau + 2^m$.
Rule 1: if $\exists\, i^* \in I$, *such that* $i^*.history = 0$ **then**
  Schedule i^*.
else
  Schedule $i^* = argmax_{\,i\in I}\ i.history$.
  Remove initial $\frac{i^*.history}{2}$ samples from i^* during update.

Algorithm 2: Multi-Scale Bootstrap DQN (MALG)

Input: n, I set of Bootstrapped DQN algorithm instances.
Initialization: Run **Algorithm 1** with inputs.
At each time step t, let the unique instance be alg, Output $\tilde{g}$ and s_t, a_t, r_t, s_{t+1}.

Non-Stationarity Detection Mechanism:

To explain the non-statioanrity detection mechanism, we first decompose the dynamic regret into two parts:

$$\sum_{\tau=1}^{t}(f_\tau^* - \tilde{f}_\tau) + \sum_{\tau=1}^{t}(\tilde{f}_\tau - R_\tau)$$

where $\tilde{f}_\tau$ is a scalar value output by the Bootstrap DQN at the beginning of each round t. This is denoted $\tilde{g}_t$ to represent the output of the active instance at time t. In a non-stationary environment, both terms can be substantially large. If we can detect the event that either of them is abnormally large, we know that the environment has changed substantially, and should just restart our base algorithm (Bootstrap DQN). To address this issue, our main idea is to maintain different instances of the Bootstrapped DQN to facilitates non-stationary detection. However, we cannot have multiple instances running and making decisions simultaneously, and here is where the optimistic estimators $\tilde{f}_\tau$'s can help. Specifically, since the quantity $U_\tau = min_{s\le\tau}\tilde{f}_s$ should always

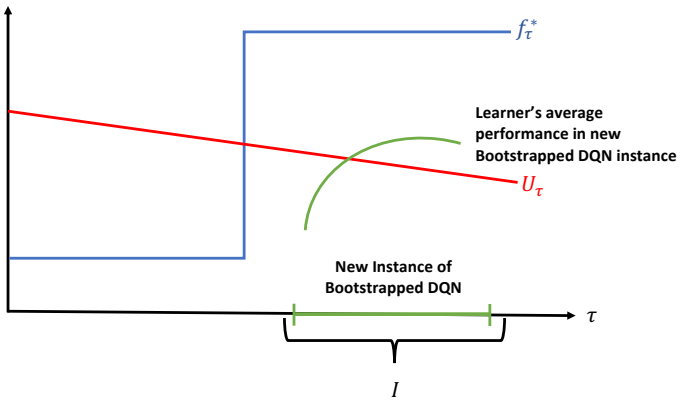

Figure 1: Non-stationary detection via multiple instances of Bootstrapped DQN

be an upper bound of the learner's performance in a stationary environment, if we find that the new instance of Bootstrapped DQN significantly outperforms this quantity at some point as illustrated in Figure 1, we can also infer that the environment has changed, and prevent the first term of the dynamic regret ($\leq \sum_{\tau=1}^{t}(f_\tau^* - U_\tau)$) from growing too large by restarting. This way the framework is able to adapt to the change by restarting its learning process to gain the new knowledge imposed by the dynamic environment.

Algorithm 3: Non-Stationary Detection and Adaptation

Initialize: $t = 1$, I set of Bootstrapped DQN algorithm instances.
for $block : n = 0, 1, ...$ **do**
  Set $t_n = t$ and initialize MALG (Algorithm 2) for the block $[t_n, t_n + 2^m]$.
  while $t < t_n + 2^m$ **do**
    Receive $\tilde{g}_t, \pi_t$ from MALG, execute π_t and receive reward R_t.
    Update MALG using $\{ s_t, a_t, r_t, s_{t+1} \}$, and set $U_t = min_{\tau \in [t_n, t]} \tilde{g}_t$.
    Perform Test 1 and Test 2. Increment $t = t + 1$. If either test fails, restart from **Line 2**.
Test 1: If $t = alg.e$ for some order m algorithm, and $\frac{1}{2^m}\sum_{t=alg.s}^{alg.e} R_t \geq U_t + \frac{\delta}{2^{m/2}}$, **return fail.**
Test 2: If $\frac{1}{t-t_n+1}\sum_{\tau=t_n}^{t}(\tilde{g}_\tau - R_\tau) \geq \frac{1}{\sqrt{t-t_n+1}}$, **return fail.**

Algorithm 3 runs MALG in a sequence of blocks with doubling lengths $(2^0, 2^1, ...)$. Within each block of length 2^m, the algorithm runs a new instance of MALG and records the minimum optimistic predictor thus far for this block $U_t = min_{\tau \in [t_n, t]} \tilde{g}_\tau$. At the end of each time, Algorithm 3 performs two tests (Test 1 and Test 2), and if either of them returns fail, it restarts from scratch.

The purpose of these two tests is to detect the event when either of dynamic regret terms is abnormally large, which will indicate that a novelty has occurred and the environment has changed. **Test 1** prevents the first term: $\sum_{\tau=1}^{t}(f_\tau^* - \tilde{g}_\tau)$ from growing too large by testing if there is some order-m instance's interval during which the learner's average performance $\frac{1}{2^m}\sum_{\tau=alg.s}^{alg.e} R_\tau$ is larger than the promised performance upper bound U_t by an amount of $\frac{\delta}{2^{m/2}}$. On the other hand, **Test 2** presents the second term: $\sum_{\tau=1}^{t}(\tilde{g}_\tau - R_\tau)$ from growing too large by directly testing if its average is large than something close to the promised regret bound $\frac{1}{\sqrt{t-t_n+1}}$.

Clearly, our algorithm doesn't require the knowledge of the degree of uncertainty of the environment, or the number of distribution changes within the environment.

Conclusion and Future Work

We studied novelty detection and adaptation in dynamic environments using Reinforcement Learning. We propose a Q-learning based framework that (1) utilizes Bootstrapped DQN for efficient and deep exploration, (2) schedules multiple instances of Bootstrapped DQN to facilitate non-stationary detection, (3) detects multiple changes in the environment by comparing the learner's average performance to the upper bound performance computed by the optimistic estimators, and (4) adapts to the detected change by restarting its learning process. Our framework achieves all of these without any knowledge of the degree of non-stationary of the environment, that is without knowing how much or how many times the distribution changes, which is often unavailable specially with highly dynamic environments.

For future work, we will implement this algorithm on different non-stationary environments such as: the Procgen Benchmark[1], which provides 16 unique environments designed to measure both sample efficiency and generalization in reinforcement learning. We will use this benchmark to compare results across different environments and evaluate the generalization of our approach.

Acknowledgments. This research is supported, in part, by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under the contract number W911NF2020003. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL, or the U.S. Government. We thank our team members on this project for all the discussions to develop this paper. Some of the ideas in this paper are based on our learning from the SAIL-ON meetings.

References

Cheung, W. C.; Simchi-Levi, D.; and Zhu, R. a. 2020. Reinforcement learning for non-stationary markov decision processes: The blessing of (more) optimism. In *International Conference on Machine Learning*, 1843–1854. PMLR.

Daniely, A.; Gonen, A.; and Shalev-Shwartz, S. 2015. Strongly adaptive online learning. In *International Conference on Machine Learning*, 1405–1411. PMLR.

[1] https://openai.com/blog/procgen-benchmark/

Domingues, O. D.; Ménard, P.; Pirotta, M.; Kaufmann, E.; and Valko, M. 2021. A kernel-based approach to non-stationary reinforcement learning in metric spaces. In *International Conference on Artificial Intelligence and Statistics*, 3538–3546. PMLR.

Efron, B.; and Tibshirani, R. J. 1994. *An introduction to the bootstrap*. CRC press.

Haliem, M.; Aggarwal, V.; and Bhargava, B. 2021. AdaPool: A Diurnal-Adaptive Fleet Management Framework Using Model-Free Deep Reinforcement Learning and Change Point Detection. *IEEE Transactions on Intelligent Transportation Systems* 1–11. doi:10.1109/TITS.2021.3109611.

Jun, K.-S.; Orabona, F.; Wright, S.; and Willett, R. 2017. Improved strongly adaptive online learning using coin betting. In *Artificial Intelligence and Statistics*, 943–951. PMLR.

Liu, Y.; Liu, Q.; Zhao, H.; Pan, Z.; and Liu, C. 2020. Adaptive quantitative trading: An imitative deep reinforcement learning approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 2128–2135.

Ortner, R.; Gajane, P.; and Auer, P. a. 2020. Variational regret bounds for reinforcement learning. In *Uncertainty in Artificial Intelligence*, 81–90. PMLR.

Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep exploration via bootstrapped DQN. *Advances in neural information processing systems* 29: 4026–4034.

Osband, I.; Russo, D.; and Van Roy, B. 2013. (More) efficient reinforcement learning via posterior sampling. *arXiv preprint arXiv:1306.0940* .

Pereira, J. 2016. Leveraging chatbots to improve self-guided learning through conversational quizzes. In *Proceedings of the fourth international conference on technological ecosystems for enhancing multiculturality*, 911–918.

Wei, C.-Y.; and Luo, H. 2021. Non-stationary Reinforcement Learning without Prior Knowledge: An Optimal Black-box Approach. *Proceedings of Machine Learning Research* 134: 1–55.