Towards Robust DeepRL: Stochastic Benchmark Environments

Olov Andersson  
Department of Computer Science  
Linköping University  
581 83, Linköping, Sweden  
olov.a.andersson@liu.se

Abstract

Deep reinforcement learning has solved a string of impressive-looking control problems in physics engines. However, it was recently shown that many of the now standard Mujoco benchmark problems are solvable by a simple linear policy [6]. This surprising result raises questions of how well RL simulation results generalize to the real world, and what would be suitable benchmarks for RL. The authors of [6] suggest widening the distribution of starting states to avoid overfitting to a “trajectory-centric” solution. We take this a step further and argue that the environments themselves should be stochastic. Autonomous robots have to deal with an uncertain world that changes over time. A second advantage is that control under inherent uncertainty is an area where traditional control approaches are difficult to apply. We introduce a seemingly simple toy example with randomly moving obstacles and try to solve it with a popular deep reinforcement learning algorithm. We find that having randomly moving obstacles makes the problem inherently non-linear, and at least an order of magnitude harder compared to both static obstacles, and surprisingly, most standard benchmarks. Further, even with 40 times as many samples as needed for the standard benchmarks, the final policy still sometimes results in inexplicable collisions. We identify tail-end convergence is an area that needs more focus for RL approaches to reach the level of robustness expected of real-world autonomous robots and vehicles.

1 Introduction

Reinforcement learning is a general approach for control and planning under uncertainty. However, the standard benchmarks for control tasks consist of a set of simulation environments in Mujoco [10] with deterministic state transitions. Simulation is a powerful tool for speeding up development, but it also carries a risk of ignoring important aspects of reality. We argue that static environments is an example of this. Real autonomous robots need to work robustly in a wide range of situations. Learning how to move from one fixed state to another is much simpler than learning what to do over a wide state distribution. In fact, in a fully deterministic environment, the optimal policy reduces to a trajectory.

Rajeswaran et al. [6] did perturbation testing of learned policies in the standard Mujoco benchmarks and found them fragile. They suggested widening the start distribution and changing the termination conditions. They further argued that in this context, the results were not impressive compared to trajectory optimization methods. We take this further and argue that environments should be stochastic, such that \( p(x_{t+1} | x_t, a_t) \) is not a deterministic function as is usually the case in modern benchmarks, even if initially unknown. The real world changes over time, and is after all not entirely predictable, that is how humans learn to generalize. Making environments stochastic is a natural way of making them more realistic, and more likely that policies will generalize to real-world problems.

Preprint. Work in progress.
Further, such stochastic control problems also offer an interesting niche that lack scalable general-purpose tools, as planning trajectories can become suboptimal. From a robotics perspective, Deep RL could perhaps be as interesting as a general-purpose way of solving control problems with inherent uncertainty, as for its model-free properties.

1.1 Deep Reinforcement Learning

We use the popular PPO algorithm, a trust-region type of policy gradient approach. Policy parameters $\theta_i$ are optimized on a surrogate surface constructed from previous trajectories via approximate importance sampling \cite{5, 7},

$$L(\theta) = \hat{E}_t \left[ \pi_{\theta_i}(a_t|s_t) \right] \left[ \pi_{\theta_{i-1}}(a_t|s_t) \right] \hat{A}_t.$$

(1)

This surrogate surface allows reusing data by running several policy gradient epochs within a trust region. In PPO this is enforced implicitly via a clipped objective controlled by $\epsilon$, see \cite{9} for details. The advantage function $\hat{A}_t$ is estimated via generalized advantage estimation (GAE) \cite{8}.

2 Introducing a Simple Benchmark with Stochastic Obstacles

To test performance on stochastic environments, we introduce a toy problem with an autonomous robot and randomly moving obstacles. We let the agent dynamics be governed by an M-dimensional Nth-order discrete-time integrator, a simplification of any system expressible as a system of linear DE’s. These are well studied in the control literature and, depending on any additional constraints, may be convex or even have simple analytic solutions. A second-order integrator acts as a pointmass under controlled forces, and e.g. quadcopter dynamics linearized around hover can be seen as a 4th or 5th order integrator. For this paper we only need a first-order integrator, defined as $x_{t+1} = x_t + 0.1a$, where in the following we assume the state is two-dimensional such that it moves in the plane. E.g. it can directly control velocity in 10Hz, which we constrain to $[-2, 2]$ m/s.

![Simple test environment. Moving obstacles in red, the agent in green. Agent preferred destination is in the black center cross.](image)

Figure 1: Simple test environment. Moving obstacles in red, the agent in green. Agent preferred destination is in the black center cross.

This undeniably simple convex problem is then augmented with up to $O=3$ spherical obstacles with 1m diameter as in Figure 1. These are given random destinations in a 3x3 square, governed by a simple P-controller with a max acceleration and velocity of 1m/s. The agent is penalized both with distance to its preferred position in the center of the square, as well as heavily penalized for running into any obstacle (or vice versa). The final cost function is chosen as

$$r(s, a) = -\|p_a - p_D\| - 500 \sum_{i=0}^{O} \text{col}(p_a, p_{oi}),$$

(2)

where $p_a$, $p_D$ and $p_{oi}$ are the positions of the agent, its destination, and obstacle $i$ respectively. The collision function $\text{col}$ is defined as the normalized penetration distance into an obstacle, where 1 is complete overlap. The penalty is chosen such that displacement from the center should always be preferable to any significant collision. With such a simple penalty, we expect that minor touches may still happen, and for real applications one may want to add some safety margin or use a more sophisticated penalty method.

The real difficulty in this environment comes from the randomly moving obstacles. Even if we here assume their current position and velocity are fully observable, their future motion is uncertain.
Inherently uncertain problems such as these require a non-linear feedback policy and often cannot be solved by classical trajectory optimization methods without approximations, e.g. [1]. Deep RL therefore offers a promising avenue for solving such problems. Here, the agent is faster than the obstacles, and it is very nimble in the simplest first-order case (it has no inertia). It should only need to plan 10-20 steps ahead and the discount factor is accordingly set to $\gamma = 0.95$. Using a higher-order integrator will impose more dynamical constraints on agent motion, also requiring harder safety considerations to efficiently account for the uncertainty.

3 Experiments

We used the PPO2 implementation from the popular OpenAI Baselines library [2]. As deep reinforcement learning can be very sensitive to hyperparameter choices [4], our strategy was to use the same normalization of state and rewards as the Mujoco defaults in Baselines, hoping to allow use of parameters close to those in [9]. After extensive experimentation with individual parameters, we found that the performance of PPO on our domain was most sensitive to step size $\alpha$, followed by the number of training epochs per batch of data. We tried adjusting $\epsilon$, which regulates the trust-region via clipping, but results were inconsistent. We also attempted to change the $\lambda$ in GAE within the suggested range of [0.9, 1.0], but it did not appear to improve final performance. Parameters used in the results were defaults for step size $\alpha = 3 \times 10^{-4}$, $\lambda = 0.95$, batch size 2048. However, 5 (vs. 10) epochs per batch was as high we could go without severe oscillations. The obstacle avoidance environment itself is implemented in OpenAI Gym and has 128 steps per episode.

3.1 The Challenge of Stochastic Environments

We test three scenarios of increasing difficulty in this domain, with five environment seeds each. We begin with a very narrow initial state distribution for the agent like in the standard Mujoco benchmarks, augmented with three static obstacles whose positions are drawn and fixed in the first episode. As seen in Figure 2a, this is very easy to solve in under the 1M steps reported for PPO on most Mujoco benchmarks [9]. In 2b we instead draw the initial agent position uniformly from the 3x3m center area like in [6], also with static obstacles. Interestingly it is only slightly slower. We suspect this is because the agent was forced to explore much of the space during training anyway, and this problem is sufficiently simple where forgetting is not an issue. Since the obstacle positions are fixed per seed, it only needs to learn to navigate this fixed world. This is verified by observing that it does not generalize well to a different environment seed. Finally, we extend this with both random start positions, and having the obstacles move randomly. We see in Figure 2c that it takes at least an order of magnitude more experience (note scale difference) to handle all the situations in this case, underscoring that it has learned a much harder problem than in the static examples. Finally, in Figure 3a we verify that contrary to some other benchmarks, a linear policy is in fact unable to solve this problem. Interestingly, with a static environment it can still attain a result that is within 100 reward from the NN policy, and just randomizing the start distribution of the agent (e.g. [6]) does not seem to be enough in this case at least.

![Figure 2](image)

Figure 2: Reward curves for learning with a) Static initial state and obstacles (0.5M steps) (b) Random initial state but static obstacles (0.5M steps) (c) Random initial state and moving obstacles (10M steps).

\(^1\)We plan to make the code public upon publication.
3.2 Convergence and Robustness

While it is easy to think it has converged at 5M steps by looking at the learning curve in Figure 2c even with policy noise set to zero, it is actually still making inexplicable choices resulting in collisions. By instead looking at log-plot of the learning curve in Figure 3b we can see that some runs may be stuck, but others are still — very slowly — converging out at 40M. The sample complexity is here getting in range of the full humanoid scenarios in [9], a vastly more complex dynamical system. Some error is expected as modern deep learning algorithms like PPO rely on several approximations, but the rate of hard collisions (>20% overlap) stubbornly remains at 0.2-0.3/min after 40M steps. We tried reducing learning rates accordingly. The occasional oscillations seen in learning curves are partially a result of tuning for as high learning speed as possible without visible impacting convergence. We tried runs up to 120M with more conservative settings on \(\alpha\), epochs, and \(\epsilon\) with only small improvements.

This does not necessarily mean that PPO cannot converge on this domain, but it requires navigating a potentially difficult interaction of four parameters governing step selection (including policy noise).

What is clear is that convergence is an aspect rarely given much attention so far. We believe that testing convergence on a diverse set of toy problems such as this has an important role in algorithm development.

![Figure 3: a) Linear policy log-learning curve for different scenarios. b) Log-plot for the NN runs from Figure 2c out to 40M steps. Runs are either still slowly converging or seemingly stuck.](image)

4 Discussion and Conclusions

In response to the recent questions raised about suitable evaluation metrics for deep reinforcement learning, we argue that uncertainty in environment transitions, in particular non-static environments like those encountered by autonomous robots and vehicles, is an important but currently overlooked aspect. To illustrate our point, we introduced a simple toy problem with randomly moving obstacles, imitating obstacle avoidance situations that arise in real-world environments. Uncertainty from obstacle motion naturally results in having to learn policies for wider state distributions. Surprisingly, this resulted in considerably slower training than reported on seemingly more impressive standard Mujoco benchmarks. We then verified that, contrary to recent results solving these with linear policies, this simple environment actually requires learning non-linear policies. We argue that such toy problems can isolate and test important properties. While not as visually impressive, they enable easier comparisons. E.g. [3] train for 64x100 CPU hours on a vastly more complicated domain with some randomization, but never precisely define the randomness, or empirically quantify the robustness of the final result.

We further discussed robustness issues we encountered using a popular method, and highlighted the importance of also analyzing tail-end convergence, and not just initial learning speed. Robustness, both for learning [4], and for the final policy, is key for many real-world applications. Learning rate plots can be deceptive. In particular they are susceptible to making choices that trade faster initial gains for poor long term convergence, or variance between runs. Perhaps as a side effect, we had difficulty getting policies to converge to sufficiently low risk of collision that it would be feasible on real-world autonomous robots, where mistakes can cause injury. As the absolute reward in learning curves can be difficult to translate into real-world performance, task-specific success criteria such as collisions may a useful complement for benchmarks. More generally, log-scale learning curves may also be useful to better highlight final policy performance.
References


