Robustness of Modern Actor-Critic Methods to Partial Observability

Stone Tejeda* stonetejeda@utexas.edu

Rishabh Rai* rishabh.rai@utexas.edu

December 9 2019

Abstract

In this work we explore how well modern actor-critic methods designed for learning in fully observable environments generalize to learning under partial observability. We consider different actor-critic algorithms, as well as different forms of partial observability and environment complexity that the agent may be presented with. Under the replicated noisy and faulty sensor conditions, results show greater difficulty when learning with noisy observations than faulty ones. We observe that, in general, SAC is more resistant to noisy conditions while TD3 is better able to perform under faulty conditions. In addition, the dimensionality of the environment appears to be inversely related to the difficulty of learning under partial observability. **Video**. **Codebase**.

1 INTRODUCTION

Within the field of Reinforcement Learning, actor-critic methods combine the strengths of value-based and policy-based methods for learning in ways that are both low-variance and maintain lenient convergence guarantees [1]. Although these methods were not specifically designed to work with occluded environments, they nevertheless have convergence guarantees that suggest robustness to missing and inaccurate features [2]. In this paper, we seek to evaluate guarantees related to partial observability by comparing the performance of three different actor-critic methods - Proximal Policy Optimization (PPO), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Soft Actor-Critic (SAC) - in both traditional and partially observable Markov decision processes.

*equal contribution

2 BACKGROUND

2.1 Partially Observable Markov Decision Processes

Traditional reinforcement learning problems model the interaction between an agent and its environment as a Markov decision process (MDP). An MDP consists of a finite set of states $\mathcal{S}$ and actions $\mathcal{A}$. It also contains two functions: a transition function $T : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ which maps state-action pairs to a distribution of future states, and a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which rewards an agent for taking a specific action in a specific state. MDPs operate in discrete timesteps, and can be either episodic (end at a terminal state $\mathcal{S}^+$) or continuous (no terminal state). At any given timestep t, an agent in the MDP framework is given the current state $s_t \in \mathcal{S}$ of the environment, and receives a reward $r_{t+1} \in \mathbb{R}$ after taking action $a_t \in \mathcal{A}$. Actions done by the agent have immediate effects on the environment, and last for only a single timestep. The overall objective for an MDP agent is to maximize future expected returns, which can be cumulative or averaged.

A partially observable Markov decision process (POMDP) is a form of MDP in which the true state of the environment is not shown to the agent [3]. Instead, the agent receives observations from a finite set of observations Ω of its world. Transition dynamics can be described by an observation function $O : \mathcal{S} \times \mathcal{A} \to \Pi(\Omega)$ which is a probability distribution over possible observations. Specifically, $O(s', a, o)$ will return the probability of making observation o, given that taking action a brought the agent to state s'. The agent, much like in an MDP, is not aware of the transition dynamics for states or observations, and receives observations and rewards from interacting with the environment. A concrete example of a POMDP would be navigating in a dark room. The true state of the agent can be described with simple cartesian coordinates, but without light, the agent can only make limited observations about the environment, such as whether it is immediately adjacent to a wall.

Agents within the POMDP framework usually store an internal belief state [3]. A belief state can be used to map observations to the most probable state the agent is in. It can also be less strict, and come in the form of probability distributions over the states of the environment. This allows for uncertainty to be encoded into the belief state. Decreasing uncertainty is a crucial component of the POMDP framework, since it is difficult to exploit an unknown state, and false beliefs can damage exploitative behavior. Uncertainty changes based on observations, which incentivizes using actions to increase the certainty of a belief state. This adds to the exploration/exploitation tradeoff, making the POMDP framework more complex than the MDP framework. Furthermore, computing the exact values for both the belief state and value function is intractable [2], so one or both must be approximated. [2] does a full survey of POMDP control methods, which include policy search, evolutionary methods, recurrent neural networks, and belief state controllers. For our work on POMDPs, we explore actor-critic methods.

2.2 Learning Methods

Methods for learning in the MDP framework fall under two main categories: value-based methods and policy-based methods [1]. Value-based methods rely on approximating the true value function of the model and typically generate policies which greedily exploit this function. More formally, value-based methods attempt to approximate a solution to the Bellman equation:

$$\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

where $p(s', r|s, a)$ is the probability of receiving reward r and being in state s' after taking action a from state s, $v_*(s)$ is the true value of state s, and γ is the discount rate.

Policy-based methods, also known as actor-only methods [1], choose instead to search for an optimal policy through a parameterized policy space. These methods typically work by estimating the gradient of the current policy's performance through interaction with the environment and update the parameters in the direction of higher performance. The policy gradient theorem provides justification for calculating the performance gradient $\nabla J(\theta)$ [4], proving that

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s, a)$$

. This theorem allows for convergence guarantees to local optima. However, since these methods don't keep track of policy histories, policy-based methods are often high variance [1].

Value and policy-based methods have complementary drawbacks. Value-based methods fail to consider the actual policy space when learning, which results in indirect learning. This indirection can lead to great value function approximations, but make no actual guarantees about the policy generated from the value function [4]. In fact, optimal policies tend to be stochastic [4], and greedy policies generated from value function approximation are determinstic, which limits the policy . Policy-based methods do consider the policy space, but fail to use previous estimates in their updates. Using previous estimates in this way is known as bootstrapping, and is usually done in value-based methods to reduce variance. Attempts to leverage these complementary issues result in actor-critic methods. Here, the critic learns a value function, and the actor updates its policy based on the critic's value function. Actor-critic methods have more lenient convergence properties while also reducing variance through bootstrapping [1]. Some of these guarantees suggest that actor-critic methods do not suffer from obscured environments [2], which shall be examined in the context of POMDPs later.

3 EXPERIMENT

3.1 Domains

In this work we focus on partially observable continuous control domains. To achieve partial observability, we alter the perceived states given by an existing fully observable environment. We specifically use the Hopper-v2 and Ant-v2 Mujoco environments implemented in OpenAI Gym. OpenAI Gym is a software library designed to abstract the environment of a reinforcement learning problem to allow for easier comparisons of RL methods [5]. Once an environment is defined within the framework, other researchers can easily re-use the abstraction with a number of other agents, as we do in this work. Mujoco is a widely used physics simulator in which a number of continuous control tasks have been implemented to act as benchmarks for control algorithm performance [6][7]. The Hopper-v2 and Ant-v2 Mujoco environments have been implemented in OpenAI Gym, and as such they are straightforward to use and modify the observations of.

Hopper-v2 is a continuous control task in which the agent must learn to control a two-dimensional one-legged robot to make it hop as fast as possible [5][7]. In this task, the full state space is observed in 11 dimensions and actions are taken in 3 dimensions. The relatively low dimensionality of this task in both state and action spaces make it one of the simpler tasks to achieve good performance in. This implementation of the Hopper benchmark is slightly different from the one presented in [7] as Hopper-v2 has 9 fewer observation dimensions, but the overall task and difficulty remain largely comparable. The other continuous control task we consider in this work is Ant-v2. Ant-v2, in which the agent must learn to control a four-legged robot to make it walk as fast as possible, is considerably larger in dimensionality than Hopper-v2, with a 111-dimensional state space and 8-dimensional action space [5][7]. The Ant implementation we use in this work lacks 4 observation dimensions when compared to the original implementation in [7], though once again remains largely comparable. We focus on these two tasks as they nearly cover the range of difficulties of the benchmark tasks presented in [7], allowing us to inspect the effects of dimensionality on robustness to partial observability.

The Hopper-v2 and Ant-v2 tasks are presented as fully observable MDPs, so to achieve partial observability we use a wrapper around the environments that partially obscures the state observations available to the learning agent. By doing so, the MDPs are transformed into POMDPs. In this work we inspect two different forms of partial observability: noisy observation space and and full obscurement of a portion of the observation space. Sensors on a real robotic agent are imperfect and may give noisy readings, so training in a simulation with noisy observations may be beneficial for Sim2Real tasks [8]. We simulate a noisy observation space by adding Gaussian white noise with standard deviation 0.1 to all dimensions of the observation space. This standard deviation of 0.1 was chosen to replicate the standard deviation chosen in [7]. Obscuring dimensions of a state space may give insight into the ability of a control algorithm to

extract latent state information from the available observations. We simulate a partially obscured ("faulty") observation space by randomly choosing (without replacement) $\lfloor 20\%\ of\ the\ number\ of\ observation\ dimensions \rfloor$ of the observation dimensions to always set to zero from the start of training onwards. In the Ant-v2 domain we limit the possible choices of dimensions to obscure to the first 27, as the dimensions afterwards are generally zero even without obscurement. Obscuring the dimensions past the first 27 would make the effects of the POMDP on learning largely trivial.

3.2 Algorithms

Twin Delayed Deep Deterministic Policy Gradient (TD3) has achieved good results in continuous control domains [9]. An extension of the Deep Deterministic Policy Gradient off-policy actor-critic algorithm, TD3 achieves superior results to its predecessor through its use of double Q learning, delayed updates, and target policy smoothing. Double Q learning mitigates the effects of maximization bias, and is extended for use in this actor-critic architecture by using the minimum of two separate critic networks to determine the value of a state-action pair:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')).$$

Delayed updates, which only update the policy once the critic has been updated enough times, reduce variance during training. Target policy smoothing is done in TD3 by adding random noise to prevent exploitation of value errors that may produce brittle policies:

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

where the noisy action is clipped to the constraints of the action space to ensure stable training. TD3 is designed specifically for continuous action spaces, which we explore in this work [9].

Soft Actor-Critic (SAC) is an off-policy continuous control algorithm combines the classical reward maximization objective with the maximum entropy objective to generate a stochastic policy:

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big) \right].$$

Here $H\left(\pi(\cdot|s_t)\right)$ is the entropy maximization augmentation, simplifying to $-\log \pi(a|s)$ in the SAC implementation. The algorithm's use of entropy maximization is particularly beneficial for exploration and resulted in good performance empirically. SAC has also been shown to be more sample efficient than prior on-policy and off-policy algorithms. The ease of hyperparameter tuning, scalability, and stability of SAC make it an attractive option for many continuous control tasks. SAC also employs the actor-critic extension of Double Q-learning that TD3 implemented [10].

Proximal Policy Optimization (PPO) is an on-policy actor-critic control algorithm that has produced good results in discrete control domains as well as in the continuous control domains that we explore here [11]. PPO extends the Trust Region Policy Optimization (TRPO) algorithm while retaining the core focus of constraining the updates that can be made to a policy in order to reduce the risk of policy collapse. TRPO accomplishes this task using a KL divergence constraint for limiting updates:

$$\theta_{k+1} = \arg\max_{\theta} \ \mathcal{L}(\theta_k, \theta) \text{s.t. } \bar{D}_{KL}(\theta||\theta_k) \leq \delta, where$$

$$\bar{D}_{KL}(\theta||\theta_k) = \mathop{\mathbb{E}}_{s\sim\pi_{\theta_k}} \left[D_{KL}\left(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s)\right)\right]$$

where θ represents the policy parameterization, $\mathcal{L}$ is the policy loss function, and δ is a predefined threshold [9]. PPO improves upon this by using a simpler clipped constraint for its updates:

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} \left[L(s, a, \theta_k, \theta)\right], where$$

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \ \ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s, a)\right)$$

where θ once again represents the policy parameterization, L is the policy loss function, A is the computed advantage, and ϵ is a predefined clipping parameter. This clipped constraint closely approximates TRPO's constraint in performance, while requiring less computation. PPO is more sample efficient, simpler to implement, and more robust to different neural network architectures than TRPO [11]. This suggests possible improvement over the TRPO results presented in [7].

The set of algorithms that we use in this work to explore robustness to partial observability consists of TD3, SAC, and PPO. We implement and apply these algorithms without modification for better performance in POMDPs. Though TD3, SAC, and PPO are not explicitly designed for training agents in partially observable environments, they have a number of desirable characteristics that lead us to believe that they will achieve superior results in the chosen domain set in comparison to other popularly used control algorithms [11][12] [10]. The fundamental actor-critic architecture shared by these three algorithms provides them an advantage over a number of other purely policy gradient-based and value function approximation-based algorithms such as REINFORCE and Q-learning, respectively. This is because actor-critic algorithms retain a convergence guarantee for local optima even in the POMDP case [2]. We chose to include PPO in the set of algorithms due to its domain flexibility well as its extensive usage in general [11]. TD3 was included because it aims to reduce maximization bias directly and has produced good results empirically [10]. TD3's use of delayed updates and policy smoothing to achieve lower variance while training also suggest the possibility of lower variance when

subjected to noisy POMDPs. SAC was included in the set of algorithms due to its stochasticity. As discussed in [3]and [1][**2**], stochastic methods are more robust to partial observability, as they may randomly take different actions in the same perceived state. This effectively mitigates the detrimental effects of perceptual aliasing, as the probability of taking the correct action for the true underlying state of a given perceived state is increased by randomness and the probability of an agent falling into a deterministic loop due to actions taken based on incomplete observations decreases [3][2]. The presence of stochasticity in SAC even when applied to MDPs also suggests that the transition to a noisy POMDP may not be as detrimental to its performance as it may be to many other deterministic control algorithms.

3.3 Design

To explore the robustness of the algorithms discussed, we compare them across the domains discussed. Each algorithms is trained in multiple versions of both the Hopper-v2 and Ant-v2 domains. For each domain, there are three different observabilities: fully observability, noisy partial observability, and obscured partial observability. The fully observable runs allow us to establish a baseline for our implementation's performance for better comparison. For consistency across algorithms and runs, the random choices for faulty partial observability are seeded with seed value of 42. Each algorithm is run for the same number of iterations and using the same hyperparameters for each benchmark as in their original papers. Detailed coverage of these parameters is available in their respective papers as well as in the codebase for this work, which can be found **here**.

4 RESULTS

The results for the runs can be seen in the Appendix. Our implementations achieved performance similar to that of their original implementations on the baseline benchmarks for each algorithm. We were unable to complete our PPO implementation due to convergence issues and time constraints, and therefore cannot draw any conclusions about PPO's performance in POMDPs. For more information regarding this, please view Future Work. In the noisy POMDP setting, SAC was able to learn a non-trivial policy for both Ant-v2 and Hopper-v2 benchmarks. TD3 was unable to do so to the same extent for Noisy Ant-v2, and was completely unable to learn in the Noisy Hopper-v2 domain. For all domains and algorithms, the agents trained in the faulty POMDP setting were able to achieve better results than those trained in the noisy POMDP settings. Unlike in the noisy POMDP setting, in the faulty POMDP setting TD3 was able to produce better results than SAC. Note that the drop in performance at the end of training for SAC faulty Ant-v2 POMDP may have been overcome if training were continued. Such recovery from drops large in SAC performance can be seen in the MDP case for the Ant-v2 benchmark. Also note that agents

trained using SAC in Ant-v2 settings were trained for 3 million iterations rather than 1 million as a result of following the number of training iterations used in the original papers for each algorithm.

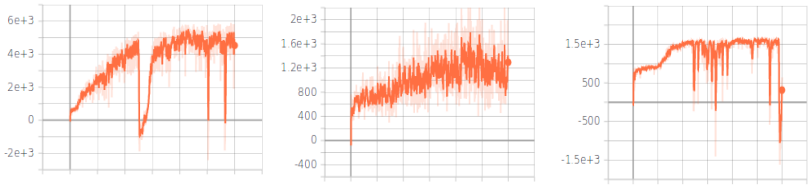
5 CONCLUSION

The varied results of the experiment can be attributed to a number of plausible factors. The overall poorer performance of the algorithms when applied to a POMDP is expected, as there is no explicit calculation of a belief state that would assist the agent in making better informed decisions. SAC's superior performance in the noisy POMDP setting can likely be attributed to its inherent stochasticity. SAC's encouragement of a more random agent during training and the use of stochastic actions make SAC resistant to noise to some extent, so adding additional zero-mean noise led to a non-catastrophic - though still noticeable - drop in performance. TD3 also employs noise during training for target policy smoothing, but its lack of entropy maximization likely makes it less resistant to additional noise than SAC. TD3's superior performance in the faulty POMDP case may be indicative of a superior ability to extract latent information from a state space. There was also a considerable difference across benchmarks. Across agents and settings, there was a greater percentage drop in performance for Hopper-v2 than for Ant-v2. We attribute this disproportionate performance drop to the smaller dimensionality of Hopper-v2 resulting in greater effect on the agent's perception of its state. A video for this project is available **here**.

6 FUTURE WORK

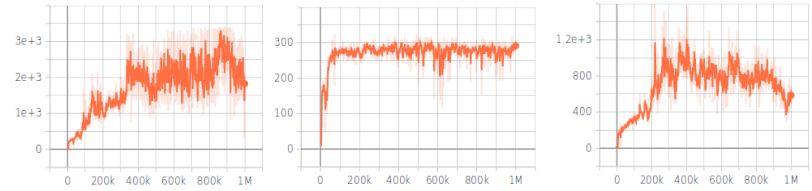
In the future we aim to fully complete the PPO implementation and the related partial observability tests. The progress on this task and an updated version of this paper that contains the new results can be found **here**.

7 APPENDIX



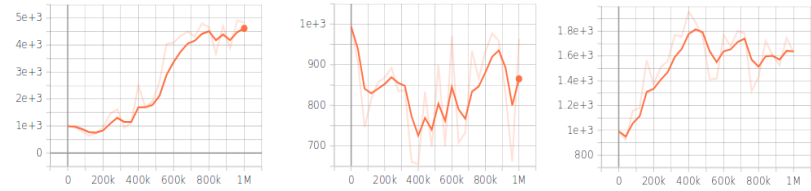
(a) MDP (b) Noisy POMDP (c) Faulty POMDP

Figure 1: Soft Actor-critic returns in the Ant-v2 environment. Note the different vertical axis scalings.



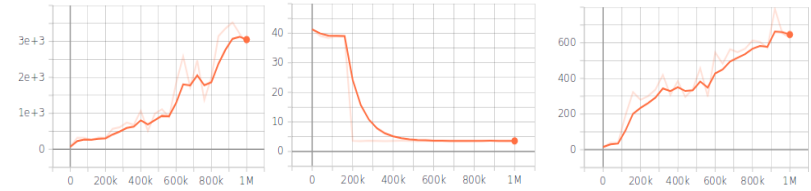
(a) MDP (b) Noisy POMDP (c) Faulty POMDP

Figure 2: Soft Actor-critic returns in the Hopper-v2 environment. Note the different vertical axis scalings.



(a) MDP (b) Noisy POMDP (c) Faulty POMDP

Figure 3: TD3 returns in the Ant-v2 environment. Note the different vertical axis scalings.



(a) MDP (b) Noisy POMDP (c) Faulty POMDP

Figure 4: TD3 returns in the Hopper-v2 environment. Note the different vertical axis scalings.

References

[1] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[2] K. P. Murphy, "A survey of pomdp solution techniques," *environment*, vol. 2, p. X3, 2000.

[3] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[4] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[6] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[7] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, pp. 1329–1338, 2016.

[8] J. Hanna and P. Stone, "Grounded action transformation for robot learning in simulation," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, February 2017.

[9] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[10] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[12] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.