remove showlabels
remove offset
set nocomments

Techniques for Sample-Efficient Reinforcement Learning

by

William Fairclough Whitney

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September, 2021

Professor Kyunghyun Cho

To my dog Weierstraß, with affection.

add dedication

Acknowledgements

- all my collaborators
- housemates

Abstract

Contents

List of Figures

List of Tables

1 | Introduction

{sec:introduction}

Reinforcement learning (RL) provides a framework for systems which learn to make decisions under uncertainty about their environment. Such a system must simultaneously determine which of all possible environments it is interacting with and solve for an optimal strategy in that environment. This uncertainty is the core challenge of reinforcement learning, and one of the most fundamental questions of the field is how much experience a learning system requires to resolve its uncertainty and perform well. We call this the sample complexity of reinforcement learning.

Classically reinforcement learning was restricted to environments with small, countable state and action spaces.[1] In this setting a variety of algorithms were developed with robust guarantees on their performance relative to the ideal policy and on the amount of data required to approach perfect behavior. Bandit algorithms for environments without temporal dependence and dynamic programming for those with non-trivial dynamics both yielded practical success. However, the specter of exponentially-increasing sample complexity limited these approaches to low-dimensional settings.

The combination of deep learning with reinforcement learning in the last decade has given rise to the new subfield of "deep reinforcement learning", which leverages function approximation to scale reinforcement learning to tasks with large or uncountable state or action spaces. Deep reinforcement learning has led to dramatic results across a range of domains, from board games like Go to complex multiplayer computer games like StarCraft, from controlling automated balloons to manipulating Rubik's cubes, and even to abstract tasks like chip design.

A key unifying feature of these most impressive deep RL results is the availability of simulators. Each of these domains admits the construction of a simulation of sufficient fidelity that a policy may be trained in simulation and then deployed on the real task with little to no fine tuning. Furthermore, these simulations are inexpensive relative to collecting "real" data, reducing the cost of training a policy by orders of magnitude. By turning data collection into computation, simulation freed deep RL from paying attention to sample efficiency, as it was more important how computationally fast an algorithm was than how many hours or years of (virtual) experience it consumed.[2]

1.1 The Cost of Free Data

Our reliance on simulation comes with hidden costs. Simulation-based RL has generated spectacular results, and simulation should be a primary tool in any effort to solve a real-world task. However, the availability of cheap simulation has shaped what the deep RL community studies by reinforcing work on simulatable domains and the large-data regime. This has resulted in slower progress on questions in the sample-limited regime. Sample-efficient reinforcement learning is important due to the fundamental scientific importance of understanding sample complexity and the intractibility of simulating every domain of interest.

Sample complexity is foundational. The study of sample complexity in reinforcement learning addresses fundamental questions about what information is needed to reliably solve a problem. The observation that realistic, high-dimensional tasks are solvable with small sample sizes may be surprising, given that there are theoretical lower bounds requiring a number of samples which is linear in the number of states or exponential in the horizon [Du et al. 2020]. This disagreement requires explanation, and it suggests that real-world problems contain a significant amount of structure which makes them easier to solve. Understanding this structure in more detail, as well as how it interacts with the inductive biases of the function approximators we use, could lead to significant breakthroughs.

Simulators are expensive. While many simulators already exist that are computationally inexpensive, many important systems of interest are computationally difficult to simulate with sufficient precision for transfer to the real world. Simulations of fluid dynamics, deformable materials, and large numbers of contacts (to name a few) can be significantly slower than real-time. Furthermore, the development of new realistic simulators is *financially* expensive not only because of the engineering of the simulator itself, but additionally due to the need to create and maintain a close correspondence between the simulation and the physical system of interest. These costs inhibit the use of simulators for domains which are too complex or too niche.

This thesis consists of work done in the last several years which makes deep RL somewhat more capable in the sample-limited regime. With this line of work I hope to unlock the wide range of environments which currently lack high-fidelity simulators. Beyond simply enabling RL in more complex environments, improvements to sample efficiency has the potential to allow agents to adapt on the fly to the specifics of the environment or objectives they interact with. A home robot might come to know the best way to pick up *your* cups, or an automated factory could produce more rapidly over the course of a production run as the networked assembly arms pool experience about manipulating each component part. In these settings learning more efficiently can be directly translated into consumer experience and dollars saved.

1.2 Elements of Sample-Efficient Learning

Reinforcement learning can be thought of as comprising two distinct processes: data collection and policy optimization. Data collection, also known as *exploration*, is when the agent interacts with the environment in order to gain more information about the optimal policy. Policy optimization is the process of using data collected from the environment to produce a policy that achieves as much reward as possible. In order that an RL algorithm overall be sample efficient, both exploration and policy optimization must themselves be efficient.

Efficient exploration means rapidly acquiring information about the optimal policy. This means collecting data that is diverse, such that valuable states and actions will be quickly uncovered, but also biasing data collection towards states and actions which are more likely to be optimal. If done well exploration spans the environment and then collects evidence to allow the agent to discard poor policies and differentiate between actions which are optimal and those which are merely good.

Efficient policy optimization means squeezing as much performance as possible out of the data which is currently available. While this is often discussed in the narrow frame of model-free RL, this process might include building models, learning representations, or even meta-learning. In principle one might hope to feed some prior beliefs along with whatever data has been collected from the environment and get back the best policy which is supported by that data. Recent work in off-policy RL and the offline RL setting have made progress towards this vision, but it remains some way off.

1.3 Overview

This thesis consists of work on improving the sample efficiency of reinforcement learning through three main directions: (1) collecting more diverse data without confounding policy learning; (2) learning representations which capture structure in the environment; and (3) studying policy optimization in the batch setting to develop policy improvement operators that are robust to limited data. The motivating challenge through much of this work is robotic manipulation using low-dimensional positions or high-dimensional images as observations and with continuous-valued action spaces. However, the findings described here are applicable more widely across reinforcement learning.

The rest of the thesis is organized as follows:

- Chapter 2 introduces background on the problem of sample-efficient RL and how it relates to the several settings under which RL has been studied.

- Part I describes the role of exploration in sample-efficient RL, with Chapter 3 illustrating how exploration techniques adapted from deep RL to the sample-limited robotic setting can improve sample efficiency and performance.

- Part II discusses the role of representation in reinforcement learning, with connections to representation learning more generally in Chapter 4 and work on representations for sample-efficient RL in Chapter 5.

- Part III studies policy optimization in the offline setting, first describing the impact of over-paraterized models in offline bandit problems in Chapter 6, then studying the repeated application of policy improvement operators in the full offline RL problem in Chapter 7.

Notes

1. Or more accurately, environments admitting assumptions that allow them to be treated as such. A 2-D continuous state space can be treated as a discrete grid of states with arbitrarily little waste under the assumption that the environment changes sufficiently slowly, for example.

2. Of course, not everyone in the deep RL community ignores sample efficiency. Notably many people working on robotics have maintained remarkable discipline in only running simulated benchmarks for physically plausible amounts of time, but there is also a small but vibrant community working on reaching human Atari performance with human-like amounts of experience.

2 | The Many Settings of Reinforcement Learning

{sec:rl-settings}

2.1 Notation

A Markov decision process (MDP) $\mathcal{M}$ consists of a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, s_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition function mapping $\mathcal{S} \times \mathcal{A}$ to distributions on $\mathcal{S}$, R is the scalar reward function on $\mathcal{S} \times \mathcal{A}$, γ is the discount factor, and s_0 is the starting state. Note that a single start state can be converted to a start distribution by letting $P(s_0, \cdot)$ be independent of the action taken. An *agent* interacts with an MDP by producing a policy π at each timestep and observing the transitions it visits.

The *value* of a state s for a policy π is the (discounted) sum of future rewards obtained by running that policy starting from the state s:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(s_t, a_t)}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\Big|\, s_0 = s \tag{2.1}$$

Similarly the value of a state-action pair (s, a) is written as

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(s_t, a_t)}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\Big|\, s_0 = s, a_0 = a \tag{2.2}$$

Any MDP admits a deterministic optimal policy π^* with corresponding value functions V^* and Q^* such that $V^*(s) \geqslant V^\pi(s)$ for all s and π [Sutton and Barto 2018].

2.2 Discounting and Resets

When $\gamma < 1$ we say that the setting is *discounted*, and for $\gamma = 1$ we say that it is *undiscounted.* An environment may have a time limit T such that after every T steps, the state is reset to s_0. In this case we call it *episodic.* To satisfy the Markov property, episodic environments should include the current timestep t in the state.

For episodic environments, it is natural to define a policy's quality in terms of the total reward earned (on average) in a single episode, with discounted environments preferring rewards obtained earlier in the episode. For non-episodic environments comparisons between policies are less clear-cut; see Section 2 of Strehl and Littman [2008] for a discussion.

Common practice in deep reinforcement learning is to train agents with discounting and resets, but not include the timestep in the state observations and simply ignore the transition from s_T to s_0. Policies are typically evaluated by the total undiscounted reward in an episode, despite the conflict with the training setup. In most cases the rewards are truncated to reflect the limited episode duration [Schulman et al. 2017; Fujimoto et al. 2018; Haarnoja et al. 2018]. Since the agent is unable to tell when an episode will end, this effectively introduces noise into value prediction targets, and this noise varies by state depending on how often the agent has ended an episode on that state. In other cases value targets may be bootstrapped from the state s_T as if the environment were not episodic. This has two issues: (1) it introduces bias by treating an estimate of $V(s_T)$ as the true value, when in some states and environments this estimate may never be updated; and (2) by pretending the environment has no resets, it introduces a mismatch between the training and test objectives.[3] However, it does not introduce noise.

More fundamentally, practically all discounted policy gradient algorithms drop the discounting term from the state distribution. Nota and Thomas [2020] show that this results in following a direction which is not the gradient of any function, and which is not guaranteed to converge to a good solution with respect to the discounted or undiscounted objectives. While this is deeply worrying, these methods frequently work well in practice. This may be due to their usage with overparameterized neural network models, which are largely invariant to a reweighting of the data [Byrd and Lipton 2018; Brandfonbrener et al. 2021].

2.3 Defining Sample Complexity

We say that a policy π is ε-optimal if $V^*(s_0) \leqslant V^\pi(s_0) + \varepsilon$. Define $\boldsymbol{\pi} = A(\mathcal{M}, i)$ to be the policy obtained by running the agent (i.e. algorithm) A in the environment $\mathcal{M}$ for i timesteps, being careful to note that the policy $\boldsymbol{\pi}$ is itself a random variable due to the randomness in the experience collected in those i steps. Let the *sample complexity* of learning a ε-optimal policy on $\mathcal{M}$ with A be the expected number of steps (indexed as i) such that

$$V^{\pi_i}(s_0) < V^*(s_0) \leqslant -\varepsilon, \qquad \text{where } \pi_i = A(\mathcal{M}, i). \tag{2.3}$$

This definition is related to those proposed by Fiechter [1994] and Strehl and Littman [2008] for PAC learning. Sometimes it is also useful to consider the "anytime" performance of an algorithm A on an MDP $\mathcal{M}$. An algorithm A_1 would dominate A_2 if $\forall i$, $V^{A_1(\mathcal{M},i)}(s_0) \geqslant V^{A_2(\mathcal{M},i)}(s_0)$.

2.4 Settings with Regret and Deployment

{sec:regret-deploymen

While the overall MDP framework is largely shared in the community, several different objectives for a learning agent are commonly studied.

The online setting. Here an RL agent learns by continually interacting with the environment with the goal of maximizing the total reward earned across all time. This gives rise to the explore-exploit tradeoff when acting: at each moment, the agent may choose to take an action which is uninformative but leads to greater short-term reward, or one which will yield more information at the cost of lower reward. The objective for this setting is to minimize the rate of accumulation of *regret*, which measures the difference between the total reward obtained by an optimal policy and the agent:

$$L(A, \mathcal{M}, T) = \mathbb{E}\left[\sum_{i=1}^{T} R(s_i^*, a_i^*) - R(s_i, a_i)\right]. \tag{2.4}$$

This setting is appropriate when an agent is being trained "on the job," where mistakes early in learning have just as much deleterious effect as those made later.

The learn-and-deploy setting. This setting consists of distinct learning and deployment phases. In the learning phase, the agent is not required to perform well and may collect whatever data is most informative. In the deployment phase, the policy is fixed and should be as close to optimal as possible. Note that the policy produced at the end of the training procedure need bear no resemblance to those used to collect data. For episodic environments, with a training period consisting of N steps we can write the this objective as

$$L(A, \mathcal{M}, N) = V^*(s_0) - V^{\pi_N}(s_0), \qquad \text{where } \pi_N = A(\mathcal{M}, N) \tag{2.5}$$

Historically this setting has not been much discussed, though it is analagous to the task of best-arm identification in bandits [Russo 2016; Kaufmann et al. 2016]. It is also related to the iterative technique of fitted Q iteration [Ernst et al. 2005; Riedmiller 2005].

Crucially, this setting is the one used in practice in nearly every application of reinforcement learning. RL algorithms are not yet safe and reliable enough to allow them to update a policy on the fly, especially with a physical system which may be damaged or cause injury. Furthermore, most works studying RL implicitly provide results in this setting by evaluating according to a different policy than the one used for training, for example showing learning curves with a deterministic policy [Mnih et al. 2015; Lillicrap et al. 2016; Fujimoto et al. 2018; Haarnoja et al. 2018]. Comparisons of final, large-data performance similarly reflect this setting [Silver et al. 2016; Vinyals et al. 2019; OpenAI et al. 2019a,b].

The offline setting. Also known as the *batch* setting, this consists only of pure policy optimization given a fixed dataset of environment interactions collected by an extrinsic behavior policy. After learning from this data in whatever way it sees fit, an algorithm produces a fixed policy with the objective of earning as much reward as possible. Though described as a reinforcement learning setting, it does not include any actual reinforcement as the agent never learns from its own interactions with the environment. However, this makes the offline RL setting uniquely valuable for isolating how much can be learned from particular data. This setting is also appealing as it would in principle allow an agent to be trained in a risk-free way by using data collected from a safe policy, and for free (in terms of samples) if data from one experiment can be repurposed as training data for another.[4]

Do I want a section on simulated versus physical envs?

Notes

3. This mismatch is decreased if the divergence between the distributions $P(s_T, \pi(\cdot \mid s_T))$ and s_0 is smaller. Like most problems in RL, it also becomes smaller if smaller discount factors are used. However in general there are actions which would provide more short-term reward, and thus perform better toward the end of an episode, than the infinite-horizon optimal actions.

4. While there are doubtless some settings where this cross-task data reuse is possible, it has quite clear limits. For a policy to be trained to solve task B using data collected from task A, the policy used for task A would have had to actually also solve task B. It could have been done piecewise rather than in a single good trajectory, but unless the tasks are extremely similar it is vanishingly unlikely. Perhaps a more practical application would be to start with a safe but poor policy for solving a task, then incrementally collect new data, refine the policy using offline RL, validate that the new policy is also safe, and then collect data once more.

Part I

Exploration and Sample Efficiency

{sec:exploration}

3 | Decoupled Exploration and Exploitation Policies

{sec:deep}

Part II

Representations and Auxiliary Tasks

{sec:representation}

4 | Evaluating Learned Representations

{sec:representation-e

As it relates to regret, don't cover the whole paper

5 | Dynamics-Aware Embeddings

{sec:dyne}

Part III

Improving Performance with Batched Data

{sec:offline}

6 | Offline Contextual Bandits with Overparameterized Models

{sec:offline-bandits}

7 | Offline RL Without Off-Policy Evaluation

{sec:offline-rl}

8 | Conclusion

{sec:conclusion}

Bibliography

Brandfonbrener, D., Whitney, W. F., Ranganath, R., and Bruna, J. (2021). Offline contextual bandits with overparameterized models.

Byrd, J. and Lipton, Z. C. (2018). What is the effect of importance weighting in deep learning? *arXiv preprint arXiv:1812.03372*.

Du, S., Kakade, S., Wang, R., and Yang, L. F. (2020). Is a good representation sufficient for sample efficient reinforcement learning? *ArXiv*, abs/1910.03016.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.*, 6:503–556.

Fiechter, C. (1994). Efficient reinforcement learning. In *COLT '94*.

Fujimoto, S., Hoof, H. V., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Kaufmann, E., Cappé, O., and Garivier, A. (2016). On the complexity of best-arm identification in multi-armed bandit models. *J. Mach. Learn. Res.*, 17:1:1–1:42.

Lillicrap, T., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Nota, C. and Thomas, P. S. (2020). Is the policy gradient a gradient? *ArXiv*, abs/1906.07073.

OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019a). Dota 2 with large scale deep reinforcement learning.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019b). Solving rubik's cube with a robot hand. *ArXiv*, abs/1910.07113.

Riedmiller, M. A. (2005). Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method. In *ECML*.

Russo, D. (2016). Simple bayesian algorithms for best arm identification. In *COLT*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. V. D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D.

(2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Vinyals, O., Babuschkin, I., Czarnecki, W., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J., Jaderberg, M., Vezhnevets, A., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5.