Searching for Simplicity and Sample Efficiency in Reinforcement Learning: Research Proposal

Trevor McInroe

Northwestern University

August 3, 2020

Research Proposal Summary

The application of reinforcement learning to real-world robotics problems is plagued with two main hurdles. The first is the combination of the difficulty of specifying an accurate reward function for complex robotic behavior, with the strong influence of the reward function on agent success. The second is the relatively slow nature of sample collection when compared to simulated environments. These issues motivate the desire for both simple, sparse reward functions and sample-efficient algorithms. However, these two goals are often in conflict, as sparse reward functions provide a minimal amount of useful feedback from which the agent is to learn. To overcome these issues, we will test Hindsight Experience Replay (HER), a replay memory designed to work specifically with sparse rewards. We will test various configurations of HER across two domains with two deep reinforcement learning algorithms.

Reinforcement Learning and Robotics

Reinforcement learning can be framed as a sequential decision making problem represented by a finite Markov decision process. An agent interacts with an environment, $\mathcal{E}$, over a series of time steps, t[1]. At each step, the agent observes a representation of the environment's state, $s \in \mathcal{S}$, and from that observation chooses an action, $a \in \mathcal{A}$. Based on the merit of the agent's action choice, a reward function produces a scalar feedback signal, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The agent's purpose is to identify an action-selection policy, $\pi : \mathcal{S} \to \mathcal{A}$, that maximizes its discounted cumulative reward over the lifetime of a given task. From these dynamics,

1. For notational brevity, current timesteps have no special marking, x, and steps one into the future are marked x'

we arrive at the following iterative *Bellman equation*, that the agent updates through its interactions:

$$Q^{\pi}(s,a) = \mathbb{E}[r + \gamma \operatorname*{argmax}_{a'} Q^{\pi}(s',a')]$$

where any arbitrary *Q-function* converges to optimality $Q^{\pi} \to Q^{*}$ as $t \to \infty$ (Sutton and Barto 2018). The Q-function can be any parameterized function, either linear or nonlinear.

Reinforcement learning has showed great success in virtual environments, where generating a large number of sample agent-environment interactions is feasible in a short amount of time. However, the application of reinforcement learning to robotics proves to not be as straightforward. In their survey work, Kober, Bagnell, and Peters (2013) outline several "curses" that plague this domain. The two that will be addressed in this work are the *curse of goal specification* and the *curse of real-world samples*.

The curse of goal specification outlines the significant role that the design of the reward function plays in determining the success of the algorithm. Oftentimes, an iterative process of defining the reward function, called *reward shaping*, is undertaken (Ng, Harada, and Russell 1999). This method can prove to be undesirable as it requires significant time and domain expertise. Alternatively, methods for learning the reward function itself have been explored, called *inverse reinforcement learning* (Abbeel and Ng 2004). In this paradigm, examples of an expert performing the task are given and the agent is to extract the implicit reward function that the expert is unknowingly maximizing. However, expert examples are not always available. Finally, the "best" reward function for a robotic agent may be complex or impossible to define due to the large number of degrees of freedom of the robot.

The curse of real world samples deals with the collection of data. Unlike simulated environments, robotic agents require interaction with the physical world, which means their movement speeds are limited by the laws of physics. Additionally, some robotic tasks may require some level of human interaction. For example, the researcher may need to "reset" the robot at the beginning state or move objects with which the robot has interacted. Thus, gathering significantly large samples of experience may be infeasible.

In combination, these two issues raise the need for (a) simple reward functions and (b) sample-efficient algorithms. The simplest reward function is a predicate, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}$, which produces a *sparse reward* that only offers informative feedback when the desired behavior is achieved. This rarely-occurring learning presents obvious issues to the performance and convergence speed of the agent. To help encourage sample efficiency, many modern reinforcement learning algorithms use a *replay memory* that stores historical tuples of transitions, (s, a, r, s'), from which mini-batches are drawn to update the Q-function. Several variations of the replay memory have been studied, such as those that can adaptively change the replay buffer size or can intelligently select which memories to keep (Liu and Zou 2017; Zilli and Hasselmo 2008). This study deals with a replay memory specifically designed to work with sparse rewards by Andrychowicz et al. (2017) called Hindsight Experience Replay (HER). HER overcomes the issues of a sparse reward by "fudging" the memories in the replay to present the outcomes as successful even though they may not be.

Experimental Design

Tasks

In this research, we will test various forms of HER across two different types of reinforcement learning problems: one with a discrete action space and one with a continuous action space. Both problems will employ a sparse reward function.

For the discrete action problem, we turn to the classic reinforcement learning environment *MouintainCar*[2], introduced by Moore (1990) in his PhD thesis. This environment presents a problem where the agent needs to drive a vehicle up a mountain to the right (see figure below). However, the vehicle's acceleration alone is not enough to get the vehicle all the way to the top. The only way to solve the problem is to first reverse the vehicle up the small slope to the left and use the momentum from that slope to get up the mountain on the right. The action space is discrete: $\{left, neutral, right\}$ and the state space is continuous, $\{velocity, position\}$.

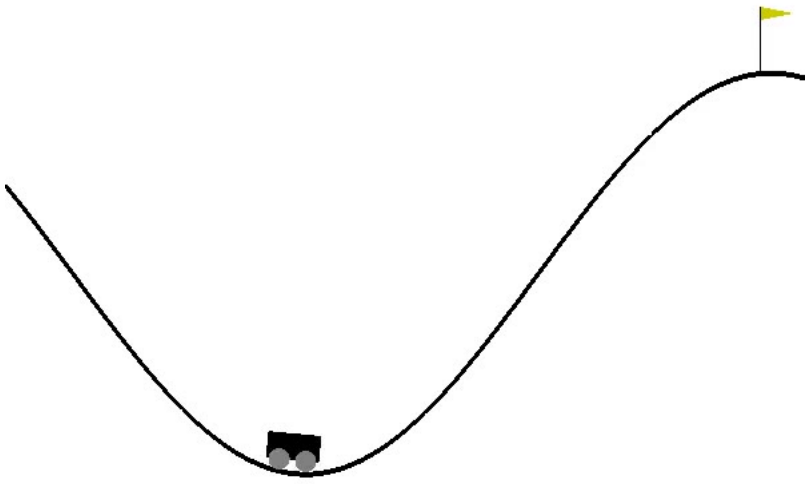

Figure 1. Depiction of the MountainCar environment.

2. https://gym.openai.com/envs/MountainCar-v0/

For the continuous action problem, we will use the continuous control version of the *LunarLander* environment[3]. This environment is meant to simulate a simple physics problem

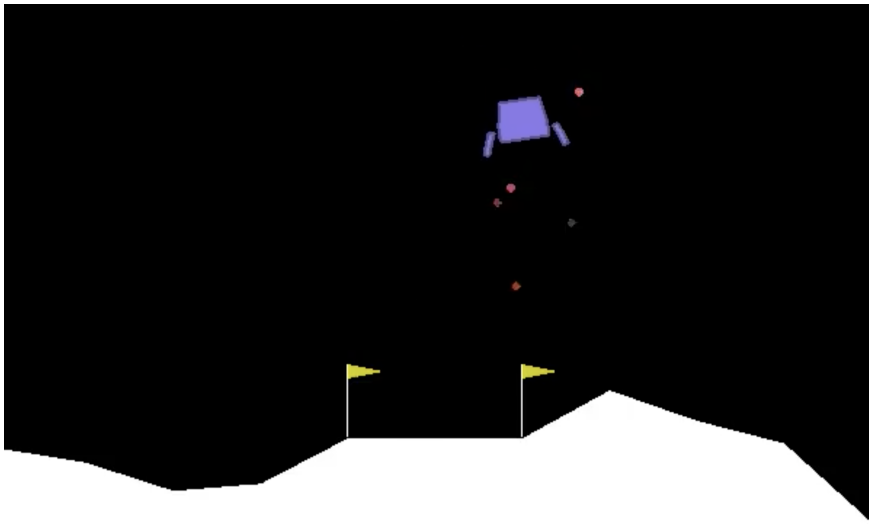

Figure 2. Image the LunarLander environment.

of gently lowering a vehicle down to a landing pad. The action-space is a 2-dimensional vector of continuous variables, $\{vertical\ thrust, horizontal\ thrust\}$, and the state space is a vector of length 8 that contains information on the lander's coordinates, speed, and angle.

Models

For this exercise, we will use two deep reinforcement learning models: Deep Q-Learning (DQN) (Mnih et al. 2015) and Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015). DQN was designed to work with discrete action spaces while DDPG is meant for continuous control problems. Both parameterize their Q-function with a neural

3. https://gym.openai.com/envs/LunarLanderContinuous-v2/

network, θ. The DQN loss function is as follows:

$$\mathcal{L}_i(\theta_i) = [y_i - Q(s, a; \theta_i)]^2$$

$$y_i = r + \gamma \operatorname*{argmax}_{a'} Q(s', a'; \theta_{i-1})$$

where the parameters at the previous step, θ_{i-1}, are a fixed snapshot during the weight update step. This leads us to the following gradient:

$$\nabla_{\theta_i} \mathcal{L}_i(\theta_i) = [r + \gamma \operatorname*{argmax}_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)] \nabla_{\theta_i} Q(s, a; \theta_i)$$

where weights are updated by drawing experience tuples from a replay memory and actions are chosen with a softmax activation as the final nonlinearity in the network.

DDPG is an actor-critic method that makes use of a collection of neural networks. The actor function, $\mu(s; \theta^\mu) + \mathcal{N}$, maps states to actions and uses a noise process, $\mathcal{N}$, to encourage agent exploration. The critic function, $Q(s, a; \theta^Q)$, "judges" the actions selected by the actor. To help with training stability, Lillicrap et al. (2015) implement target networks for both the critic, $Q'(s, a; \theta^{Q'})$, and actor, $\mu'(s; \theta^{\mu'})$, that slowly track the weights of their tied networks. This is all brought together during the weight updating step using a mini-batch

drawn from the replay memory:

$$\nabla_{\theta_i^Q}\mathcal{L}_i(\theta_i^Q) = [r + \gamma Q'(s', \mu'(s'; \theta^{\mu'}); \theta^{Q'}) - Q(s, a; \theta^Q)]\nabla_{\theta_i^Q}Q(s, a; \theta^Q) \tag{1}$$

$$\nabla_{\theta^\mu}\mathcal{L}_i(\theta^\mu) = \nabla_a Q(s, a; \theta^Q)\nabla_{\theta^\mu}\mu(s; \theta^\mu) \tag{2}$$

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'} \tag{3}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'} \tag{4}$$

The critic is updated with the gradient in Equation (1), the actor is updated with the gradient in Equation (2), and the tracking networks are updated in Equations (3) and (4). For a graphical depiction of the generic actor-critic design, see the Figure 3, below.

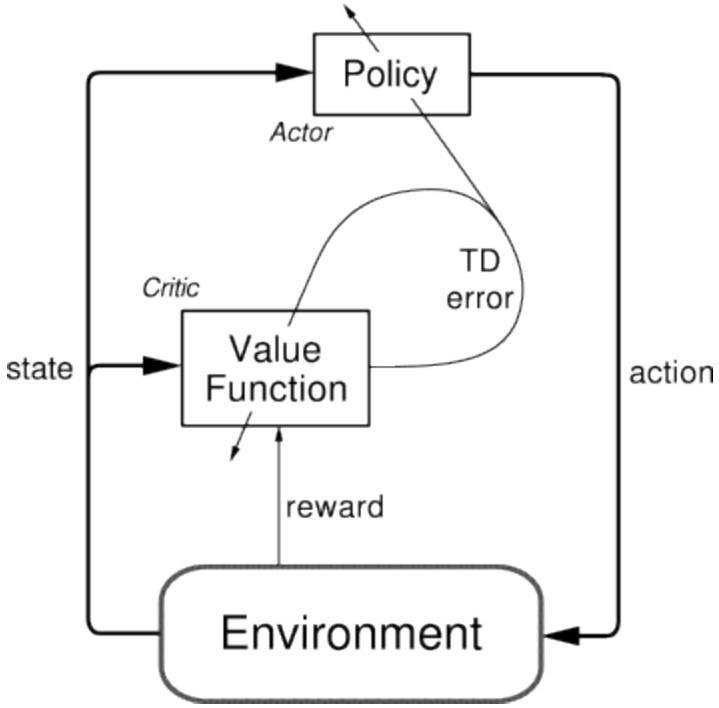

Figure 3. Depiction of actor-critic process from Sutton and Barto (2018)

Bibliography

Abbeel, Pieter, and Andrew Y. Ng. 2004. "Apprenticeship Learning via Inverse Reinforcement Learning". In *Proceedings of the Twenty-First International Conference on Machine Learning*, 1. Banff, Alberta, Canada: ACM.

Andrychowicz, Marcin, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. 2017. "Hindsight Experience Replay". In *31st Conference on Neural Information Processing Systems*. Long Beach, CA: ACM.

Kober, Jens, J. Andrew Bagnell, and Jan Peters. 2013. "Reinforcement Learning in Robotics: A Survey". *The International Journal of Robotics Research* (London, England) 32 (11): 1238–1274.

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. "Continuous Control with Deep Reinforcement Learning". arXiv: `1509.02971[cs.LG]`.

Liu, Ruishan, and James Zou. 2017. "The Effects of Memory Replay in Reinforcement Learning". arXiv: `1710.06574[cs.AI]`.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2015. "Human-Level Control Through Deep Reinforcement Learning". *Nature* 518.

Moore, Andre. 1990. "Efficient Memory-Based Learning for Robot Control". PhD thesis, University of Cambridge.

Ng, Andrew, Daishi Harada, and Stuart Russell. 1999. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In *Proceedings of the Sixteenth International Conference on Machine Learning*, 278–287. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press.

Zilli, Eric, and Michael Hasselmo. 2008. "The Influence of Markov Decision Process Structure on the Possible Strategic Use of Working Memory and Episodic Memory". *PLoS One* 3 (7).