A Novel Benchmark Methodology and Data Repository for Real-life Reinforcement Learning

Ali Nouri, Michael L. Littman, Lihong Li ⋆ {NOURI,MLITTMAN,LIHONG}@CS.RUTGERS.EDU

Ronald Parr, Christopher Painter-Wakefield, Gavin Taylor† {PARR,PAINT007,GVTAYLOR}@CS.DUKE.EDU

⋆ Department of Computer Science, Rutgers University, Piscataway, NJ 08854 USA

† Department of Computer Science, Duke University, Durham, NC 27708 USA

Although reinforcement learning (RL) defines the learning problem in a more general setting than does supervised learning, making it a better fit to a broader spectrum of real-life learning tasks, it has been far less successful in gaining attention from other scientific disciplines and commercial ventures beyond machine learning. In an effort to help coax reinforcement learning out of the labs, we introduce a novel benchmark methodology and problem repository. Our approach involves collection and distribution of static training/testing datasets, similar to efforts in other disciplines such as the UCI repository for supervised learning [4]. The new framework allows for a fresh set of benchmarks grounded in measured data that are challenging, consistent, and realistic. We believe that the new approach can succeed as a focal point for empirical developments in the field.

The current situation in reinforcement learning is akin to the pre-UCI era in supervised learning. New algorithms are validated with one or two artificial benchmark problems and often require problem-specific tweaking. Recently however, some impressive attempts have been made to grow reinforcement learning out of this model. The RL repository at the University of Massachusetts [1], the University of Alberta's RL-Glue project [2] and the RL Competition are examples that have helped unify the evaluation of RL algorithms. However, the nature of the evaluation schemes of these methods have made it difficult to include actual measured data from real-life domains, like those found in the supervised-learning benchmarks.

Creating an RL benchmark is not straightforward. For example, it is not apparent what data format should be used. In supervised learning, data can easily be provided as text files, usually in the form of a mapping from input space X to an output space Y. However, RL environments are typically viewed as "live" objects that respond dynamically to action choices by an agent, making it difficult to represent them by flat files. The natural solution of providing environments as executable programs effectively limits the field to domains that are simulations.

In this work, we consider the problem of offline policy evaluation with batch data. Since policy evaluation is an important building block of a wide class of reinforcement-learning algorithms [3, 2], we believe the new framework can make a two-fold contribution to the field: First, practitioners looking for the best solution to challenging real-life problems can post their data to the database to allow the community to evaluate different algorithms on the data; second, algorithm designers can test their methods on real, measured data instead of toy domains, without having to create and maintain their own physical or virtual environments.

Below, we briefly describe the methodology from the standpoint of the end-user and leave the theoretical analysis to a more detailed paper.

Objective. The goal of the learning process is to estimate the function $Q^\pi : S \times A \rightarrow \Re$ for a fixed policy π. Function $Q^\pi(s,a)$ is defined as the expected sum of discounted rewards the agent collects, when it starts in state s, executes action a, and then follows policy π thereafter.

A learning algorithm takes in a set of training data consisting of traces collected from a dynamical system, and has to output $\widehat{Q^\pi}$ as an estimation of Q^π for a set of testing state-action pairs.

A trajectory, denoted by ρ, is a sequence of transitions $\{t_1, \ldots, t_l\}$, where each t_i is a tuple $t = \langle s, a, \pi(s), r, s', \pi(s') \rangle$. We denote the i-th transition by $\rho(i)$ and interpret it as the following: From state s, the agent took action a, went to state s' and received the reward r. The two fields $\pi(s)$ and $\pi(s')$ correspond to the actions that policy π suggested for the two states s and

[1] http://www-anw.cs.umass.edu/rlr/

[2] http://glue.rl-community.org/

s'. Let π_e be an auxiliary policy intended to have a fast mixing rate. Intuitively, this policy is used to perform the task of resetting the environment to avoid collection of biased data [2].

To collect data from an environment, the two policies π and π_e are executed in the following way: Start a trajectory ρ_i by executing one random action, followed by the execution of actions using π. End the trajectory with probability γ at each timestep, where γ is the discount factor. Once the trajectory has terminated, execute π_e for a fixed number of timesteps to allow the environment to mix. Tag the data from the execution of π_e as exploratory and return to collecting data from π as above.

A set of trajectories can be partitioned into training and testing using standard techniques from the supervised-learning community, such as leave-one-out cross validation. However, an important distinction from supervised datasets is that the labels for the testing set are not readily available. Since there is no access to the true value of Q^π, a rollout technique is used to approximate it by $\widetilde{Q^\pi}$. Formally:

$$\widetilde{Q^\pi}(\rho(1).s, \rho(1).a) = \sum_{i=1}^{l} \rho(i).r, \qquad (1)$$

where ρ is a testing trajectory and l is its size. Using this approach, only the first element of each testing trajectory is considered as testing point, and the rest are used to compute the label. This process ensures that we have a less biased set of testing points than if every step on a trajectory of π were used.

As a motivating example, we describe an example application of the database, in which data collected from a real robot is used to evaluate parameter settings of an algorithm. Specifically, we consider the LSTDQ algorithm [3] with tile coding [5] as the basis function, and look at the effect of the discretization level of the tile coding on the final performance. The data we used comes from a robotic extension of the PuddleWorld environment [1]. Here, a Sony Aibo robot was put in an enclosed box and had to go to one of the corners while avoiding two imaginary puddle regions. The state space consists of the robot's X and Y coordinates and its orientation, and the available actions are $\{$forward, backward, turn left/right, strafe left/right$\}$. Each step in the environment costs 1 as long as the robot is outside the puddles. Entering puddles costs 40 and each time the robot goes to the goal region, it receives $+20$ and randomly moves to a new location.

A simple hand-designed policy that tries to follow the shortest path to the goal was used as the policy π. The exploratory policy, π_e, selected actions using a uniform,

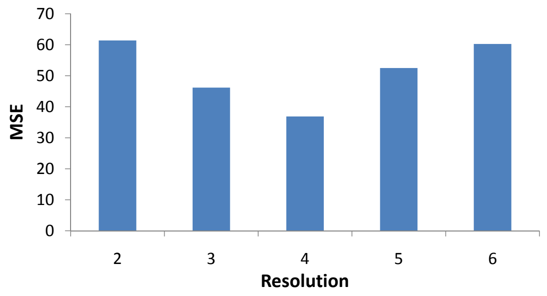

Figure 1. Using data from a robotic task to fine-tune the parameters of the LSTDQ algorithm.

random distribution over actions. A dataset of 500 trajectories was gathered and we used it to generate Figure 1. For each learning pass, the dataset was randomly partitioned into two equally-sized training and testing trajectories. The error function is the mean squared error (MSE) between $\widetilde{Q^\pi}$ and $\widehat{Q^\pi}$. Finally, the result was averaged over 20 trials for each of the tile resolutions, which varied from 2 to 6 per dimension.

While crude discretizations are not expressive enough to represent the Q^π function, yielding high errors due to the large bias of the function approximator, fine resolution results in high errors simply because not enough training data is available for each cell—an error due to high bias. Middle values achieve the best balance between bias and variance (see figure).

Although our data repository ignores action selection and exploration—central problems to the reinforcement-learning framework—we believe that there is value in providing a resource that allows users to focus on the problem of learning and generalization on measured data. We plan to involve the community in expanding this resource in the years to come.

References

[1] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *NIPS '05*, pages 369–376, Cambridge, MA, 1995. The MIT Press.

[2] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML '02*, pages 267–274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[3] Michail Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)*, 4:1107–1149, 2003.

[4] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.