Journal of Machine Learning Research x (2015) x-xx Submitted 11/13; Revised 11/14; Published 01/15

RLPy: A Value-Function-Based Reinforcement Learning Framework for Education and Research

Alborz Geramifard[12] AGF@CSAIL.MIT.EDU
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology,
77 Massachusetts Ave., Cambridge, MA 02139 – USA
Christoph Dann[1] CDANN@CMU.EDU
Machine Learning Department, Carnegie Mellon University,
5000 Forbes Ave., Pittsburgh, PA 15213 – USA
Robert H. Klein[1] BOBKLEIN2@ALUM.MIT.EDU
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology,
77 Massachusetts Ave., Cambridge, MA 02139 – USA
William Dabney[2] WDDABNEY@AMAZON.COM
Amazon.com,
440 Terry Ave. N, Seattle, WA 98109 – USA
Jonathan P. How JHOW@MIT.EDU
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology,
77 Massachusetts Ave., Cambridge, MA 02139 – USA

Editor: Geoff Holmes

Abstract

RLPy is an object-oriented reinforcement learning software package with focus on value-function-based methods using linear function approximation and discrete actions. The framework was designed for both education and research purposes. It provides a rich library of fine-grained, easily exchangeable components for learning agents (e.g., policies or representations of value functions), facilitating recent increased specialization in reinforcement learning. RLPy is written in Python to allow fast prototyping but is also suitable for large-scale experiments through its inbuilt support for optimized numerical libraries and parallelization. Code profiling, domain visualizations, and data analysis are integrated in a self-contained package available under the Modified BSD License at http://acl.mit.edu/rlpy. All these properties allow users to compare various reinforcement learning algorithms with little effort.

Keywords: reinforcement learning, value-function, empirical evaluation, open source

1. Introduction

An integral part of most introductory artificial intelligence courses are value-function-based methods using linear function approximation for solving Markov decision processes (such as

1. The first three authors contributed equally to this work.
2. The majority of this work was done prior to Amazon involvement of the authors. This paper does not reflect the views of the Amazon company.

linear Q-learning or SARSA). In addition, many researchers build upon this well-understood and powerful framework and aim at improving existing methods by, for example, feature learning (Keller et al., 2006; Parr et al., 2007; Geramifard et al., 2011), or policies that better trade off exploration and exploitation (for example Nouri and Littman, 2009; Jaksch et al., 2010; Li, 2012).

The need to unify and increase reusability of software packages for reinforcement learning research has been widely discussed (Tanner and White, 2009; Schaul et al., 2010), and many successful tools have been created (Tanner and White, 2009; Schaul et al., 2010; Degris, 2013; de Comite, 2006). However, it is desirable to have a software framework that is 1) easily accessible by novices so that they may compare and understand existing algorithms, and 2) efficient for researchers who desire to perform large scale experiments and advance the state-of-the-art.

By focusing on the prominent class of value-function-based methods with linear function approximation using discrete actions, RLPy aims at being such a software framework that provides simple and convenient tools for conducting sequential decision making experiments. In the following, we present the main features of RLPy and highlight those that distinguish it from existing frameworks.

2. Existing Frameworks

The following existing software packages have some overlap with RLPy:

1. *RL-Toolbox: (Neumann, 2005)* C++ RL toolbox focusing on continuous state-spaces
2. *CLSquare: (Riedmiller et al., 2012)* C++ RL framework focused on interfaces with several robotics
3. *libPG: (Aberdeen, 2007)* RL library focused on high-performance policy-gradient algorithm implementations
4. *rllib (Frezza-Buet and Geist, 2013):* Template-based C++ RL library for value-function methods
5. *rl-texplore-ros-pkg:(Hester, 2013)* ROS package for RL algorithms
6. *JRLF: (Kochenderfer, 2006)* Small-scale Java-Framework for RL experiments
7. *PIQLE: (de Comite, 2006)* Java-Framework for RL experiments
8. *RLPark:(Degris, 2013)* Java reinforcement learning library
9. *RLLib: (Abeyruwan, 2013)* Port of RLPark into C++
10. *RL-Glue, RL-Library: (Tanner and White, 2009)* Protocol for RL experiments and reference implementations
11. *ApproxRL: (Busoniu, 2010)* Matlab Toolbox with RL and dynamic programming algorithms
12. *MMLF: (Metzen and Edgington, 2011)* Python-based framework for reinforcement learning
13. *PyBrain (Schaul et al., 2010):* Machine learning library focused on neural networks with RL support

For the sake of brevity, we do not compare RLPy against each of the existing frameworks in detail but highlight key differences in the following section by referencing the list above.

3. Why RLPy?

Improved Granularity of Agents with Linear Value Functions. RL has advanced significantly over the past decade, leading researchers to narrow their focus towards specialized, independent aspects of RL agents, such as approximate function representations, exploration schemes, and learning rates. The structure of numerous existing frameworks (2, 6, 7, 10, 12) does not properly account for this increased specialization and makes it cumbersome to exchange, for example, the way the value function is represented in a learning agent. RLPy addresses this issue by separating these components into exchangeable classes (shown as green boxes in Figure 1) and other minor components such as learning rates into separate

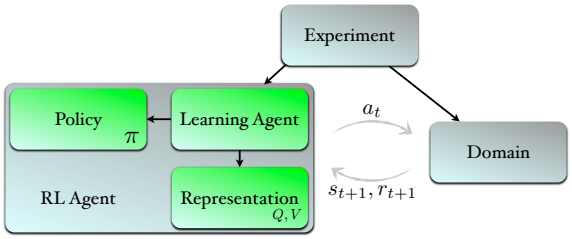

Figure 1: RLPy framework - Green components constitute an RL agent which did not exist as separate components in previous RL frameworks. The experiment module handles the interaction between the agent and the domain; gray arrows depict the information flow in a conventional RL framework (See e.g. Sutton and Barto, 1998).

```
import rlpy
#### Domain ####
domain = rlpy.Domains.InfCartPoleBalance()
### Agent ####
representation  = rlpy.Representations.Tabular(domain, discretization=20)
policy = rlpy.Policies.eGreedy(representation, epsilon=0.1)
agent = rlpy.Agents.SARSA(policy, representation, domain.discount_factor)
### Experiment ####
experiment = rlpy.Experiments.Experiment(agent, domain, max_steps=100)
experiment.run()
experiment.save()
```

Figure 2: RLPy code for setting up and running an experiment: SARSA learning for $100,000$ steps how to balance an inverted pole on a cart while following an ϵ-greedy policy. The representation is a discretized table lookup with 20 bins spaced uniformly in each dimension.

functions. This division reduces implementation effort, promotes reusability, and facilitates automated testing. Code for an example experiment that exploits this modularity is shown in Figure 2. In addition, the assumption of linearly parameterizing the value function allows RLPy to provide many tools and helpers for designing state features. For example, in large MDPs where using a tabular representation is infeasible, the `IndependentDiscretization` representation creates a set features by ignoring dependency among dimensions of the state space. Further dependency between such features can be added to the representation using the `iFDD (incremental feature dependency discovery)` module.

Rapid Prototyping with Python. RLPy is fully object-oriented and based primarily on the Python language (van Rossum and de Boer, 1991). Low-level, computationally-intensive tools are implemented in Cython (a compiled and typed version of Python) or C++. In contrast to other packages (1 – 9) written solely in C++ or Java, this approach leverages the user-friendliness, conciseness, and portability of Python while supplying computational efficiency where needed. This combination allows researchers to prototype new ideas quickly and comfortably without sacrificing the computing speed necessary to conduct large-scale experiments. In addition, the Python-based approach of RLPy is particularly suited for education as it does not require any proprietary software (in contrast to 11).

"Batteries Included" – Many Existing Components and Benchmarks. RLPy includes an ever-growing repository of components which may be combined to form new RL agents. While many frameworks (1, 3, 4, 6) only include classic benchmark domains such as *PuddleWorld* or an *Inverted Pendulum on Cart*, RLPy supplies a large number of more challenging domains such as *HIV-Treatment*, *Hovering a Helicopter*, and *Pac-Man*. In addition to implementations of most value-function-based RL algorithms, RLPy includes experimental support for dynamic programming methods that require full domain knowledge but yield optimal policies. This is especially useful as a baseline for comparison with (often sub-optimal) policies generated by RL agents.

Ease of Use and Development. Numerous tools are shipped with RLPy that facilitate ease of use and efficiency. One example is the code profiler, which produces a visual runtime graph of the source code (c.f. Figure 3 right) and identifies slow routines. This information allows the researcher to reduce the runtime of an algorithm with minimal effort and discourages premature runtime optimization. Additionally, every RLPy domain has a visualization, an important feature lacking in other frameworks (3, 4, 7). These visuals help students and researchers quickly assess and gain intuition about the algorithm and domain behavior.

Automation of Experiments RLPy aims to promote reproducible research. To this end, it provides a suite of tools to automate the entire experiment pipeline. For example, RLPy allows concise specification of experiment settings (see Figure 2) and automated and efficient hyperparameter optimization with the `hyperopt` package (Yamins et al., 2013). Researchers can share their experimental setups by publishing short settings files, and colleagues can reproduce the results when running the scripts independent of their hardware or operating system. Additionally, RLPy experiments are natively parallelizeable. Once parameters are selected, the user simply specifies the number of CPU cores RLPy can utilize for multiple experiments to test statistical significance. RLPy enables further scaling by switching seamlessly from a single machine to a job-based cluster (e.g. HTCondor) while ensuring results remain identical across varying hardware. RLPy also provides automated tools for generation of final publication-ready plots of results (see Figure 3 left); researchers need only specify the quantities that should appear on the plot. To the best of our knowledge this degree of automation of the entire experimentation pipeline is unique to RLPy.

4. Conclusion

RLPy is a new reinforcement learning framework focused on value-function-based reinforcement learning using linear function approximation with discrete actions. It reflects recent developments in the field. RLPy simplifies the construction of learning agents and makes it easier for novices and experts alike to evaluate and compare algorithms, representations, environments, and other RL components and modules, and to construct their own. It also provides many tools for conducting reproducible experiments from initial prototyping to final plotting. The framework is entirely open-source and all contributions are welcome and encouraged.

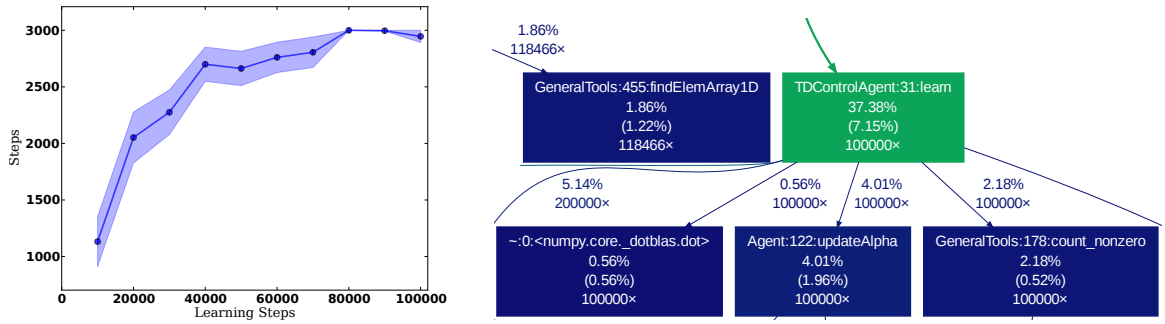

Figure 3: RLPy sample outputs of RLPy plotting (left) and profiling (right) tools: A portion of the profiling graph of the example code (Figure 2) in which the green box shows the statistics of executing the `learn` function 10^5 times. It required 37.38% of the CPU-time for completion, out of which its main body was responsible only for 7.15% of the computation while the rest was spent in other called functions (shown as output arrows).

Acknowledgments: Many thanks to all those who helped develop and test RLPy in its early stages, especially David E. Williams. Thanks also to the ever-growing body of students and researchers who continue to improve and contribute to the project.

References

Douglas Aberdeen. LibPGRL: A high performance reinforcement learning library in C++, 2007. URL https://code.google.com/p/libpgrl.

Saminda Abeyruwan. RLLib Lightweight Standard and On/Off Policy Reinforcement Learning Library (C++), 2013. URL http://web.cs.miami.edu/home/saminda/rllib.html.

Lucian Busoniu. ApproxRL: A Matlab Toolbox for Approximate RL and DP, 2010. URL http://busoniu.net/files/repository/readme_approxrl.html.

Francesco de Comite. PIQLE: A Platform for Implementation of Q-Learning Experiments, 2006. URL http://piqle.sourceforge.net.

Thomas Degris. RLPark, 2013. URL http://rlpark.github.io.

Herve Frezza-Buet and Matthieu Geist. A C++ Template-Based Reinforcement Learning Library : Fitting the Code to the Mathematics. *Journal of Machine Learning Research*, 14:625–628, 2013.

Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan How. Online discovery of feature dependencies. In Lise Getoor and Tobias Scheffer, editors, *International Conference on Machine Learning (ICML)*, pages 881–888. ACM, June 2011. ISBN 978-1-4503-0619-5.

Todd Hester. rl-texplore-ros-pkg: Reinforcement learning framework, agents, and environments with ROS interface, 2013. URL https://code.google.com/p/rl-texplore-ros-pkg.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 11:1563–1600, 2010.

Philipp W. Keller, Shie Mannor, and Doina Precup. Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, 2006.

Mykel Kochenderfer. JRLF: Java Reinforcement Learning Framework, 2006. URL http://mykel.kochenderfer.com/jrlf.

Lihong Li. Sample complexity bounds of exploration. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State of the Art*. Springer Verlag, 2012.

Jan Hendrik Metzen and Mark Edgington. Maja machine learning framework, 2011. URL http://mmlf.sourceforge.net.

Gerhard Neumann. *The Reinforcement Learning Toolbox , Reinforcement Learning for Optimal Control Tasks*. PhD thesis, TU Graz, 2005.

Ali Nouri and Michael L. Littman. Multi-resolution exploration in continuous spaces. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 1209–1216. MIT Press, 2009.

Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael Littman. Analyzing Feature Generation for Value-Function Approximation. In *International Conference on Machine Learning (ICML)*, 2007.

Martin Riedmiller, Manuel Blum, and Thomas Lampe. CLS^2: Closed Loop Simulation System, 2012. URL http://ml.informatik.uni-freiburg.de/research/clsquare.

Tom Schaul, Justin Bayer, Daan Wierstra, Yi Shun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. PyBrain. *Journal of Machine Learning Research (JMLR)*, 11:743–746, 2010.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, volume 9 of *Adaptive computation and machine learning*. MIT Press, 1998. ISBN 9780262193986.

Brian Tanner and Adam White. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research (JMLR)*, 10:2133–2136, September 2009.

Guido van Rossum and Jelke de Boer. Interactively testing remote servers using the python programming language. *CWI Quarterly*, 4(4):283–303, december 1991. Amsterdam.

Daniel Yamins, David Tax, and James S. Bergstra. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David Mcallester, editors, *International Conference on Machine Learning (ICML)*, volume 28, pages 115–123. JMLR Workshop and Conference Proceedings, 2013.