RLzoo: A Comprehensive and Adaptive Reinforcement Learning Library

Zihan Ding
Imperial College London
zd2418@ic.ac.uk

Tianyang Yu
Nanchang University
tianyang2017@gmail.com

Yanhua Huang
Xiaohongshu Technology
iofficium@gmail.com

Hongming Zhang
Peking University
zhanghongming@pku.edu.cn

Luo Mai
University of Edinburgh
luo.mai@ed.ac.uk

Hao Dong
Peking University
Peng Cheng Laboratory
hao.dong@pku.edu.cn

September 21, 2020

Abstract

Recently, we have seen a rapidly growing adoption of Deep Reinforcement Learning (DRL) technologies. Fully achieving the promise of these technologies in practice is, however, extremely difficult. Users have to invest tremendous efforts in building DRL agents, incorporating the agents into various external training environments, and tuning agent implementation/hyper-parameters so that they can reproduce state-of-the-art (SOTA) performance. In this paper, we propose RLzoo, a new DRL library that aims to make it easy to develop and reproduce DRL algorithms. RLzoo has both high-level APIs and low-level APIs, useful for constructing and customising DRL agents, respectively. It has an adaptive agent construction algorithm that can automatically integrate custom RLzoo agents into various external training environments. To help reproduce the results of SOTA algorithms, RLzoo provides rich reference DRL algorithm implementations and effective hyper-parameter settings. Extensive evaluation results show that RLzoo not only outperforms existing DRL libraries in its simplicity of API design; but also provides the largest number of reference DRL algorithm implementations.

Keywords Reinforcement Learning, Programming Abstraction, Hyper-parameters

1 Introduction

In the last few years, we have seen many successful applications of Deep Reinforcement Learning (DRL) technologies in different sectors, such as computer gaming [1, 2], robotic control [3, 4], self-driving cars [5], language modeling [6, 7] and optimisation [8] . To adopt these technologies, DRL developers often need to construct and evaluate various DRL agents. These agents implement different DRL algorithms, e.g., on-policy [9, 10] and off-policy [1]. They interact with external training environments, e.g., OpenAI Gym [11] and RLbench [12] to evaluate their performance, usually through reward functions. They iteratively update algorithm parameters (e.g., model weights) until they can achieve target rewards.

Though promising, fully achieving the promise of DRL in practice is extremely difficult. DRL practitioners often report three major challenges. First of all, DRL agents are difficult to build. Developers face many different DRL algorithms that are potentially useful for their applications. They have to prototype and evaluate many candidate DRL algorithms and find the best-performing one, which is time-consuming. In addition, DRL agents must interact with different external environments. These environments produce various kinds of data (e.g., vector, image, and dictionary).

Developers must *manually* modify the DRL agents and integrate them into external environments. This integration step requires expertise in DRL algorithm, which is unfortunately not available in most DRL users. Last but not least, DRL developers often find it difficult to reproduce state-of-the-art (SOTA) result. The performance of a DRL agent is sensitive to its implementation and chosen hyper-parameters. Developers have to spend tremendous resources in tuning their agents' implementation and hyper-parameters.

Existing DRL libraries, however, cannot fully address the above requirements. DRL libraries, including OpenAI Baselines [13], Stable Baselines and Coach [14], provide rich pre-defined DRL agents for classical benchmarks. These libraries pursue simplicity and provide only command-line interfaces. This however prevents their users from (i) integrating their agents into practical applications, and (ii) customising the agents for better performance. Their demonstration purpose also makes them lack of advanced DRL environments, e.g., RLbench [12], and useful distributed DRL algorithms like in [15].

Alternatively, users could use programmable DRL libraries including Tianshou [16], keras-rl [17], and Tensorforce [18]. These libraries are designed for DRL researchers. They expose low-level programming APIs to help researchers invent DRL technologies. They however become ill-suited for users who may not have the comprehensive knowledge of DRL. Their focus of supporting researchers also make them lack of a wide range of pre-defined DRL environments and algorithms.

In this paper, we introduce RLzoo, a library that helps users to develop, customise, and explore DRL technologies. The design of RLzoo makes the following contributions:

(i) Expressive and flexible DRL agent APIs. RLzoo proposes both high-level APIs and low-level APIs. The high-level APIs can facilitate users in developing complex DRL agents; while the low-level APIs allow users to easily customise the agents for best possible agent performance. This setting makes it easy to prototype DRL agents. A DRL agent can be declared in a concise manner, following four abstracted steps (each step corresponds to a single API): declaring an environment, choosing a DRL algorithm, building a DRL agent, and launching the agent for training. This high degree of abstraction does not compromise flexibility. RLzoo exposes a clean interface in its APIs to take custom components for the agents, such as custom neural networks, optimisers, and hyper-parameters.

(ii) Adaptive agent construction algorithm. Supporting numerous custom components in agents introduces challenges. Users would have to manually re-configure many agent components during the re-configuration process, as in existing DRL libraries. To avoid expensive manual re-configuration, we achieve adaptive agent construction in RLzoo. RLzoo proposes an adaptive agent construction algorithm that can automatically integrate custom DRL agents into different external training environments. We design different *adaptors* and place them in between agent components. These adaptors can infer any component change in the agent, e.g., importing a new environment or a custom agent component. They will automatically propagate the corresponding component update to handle the change within the agent.

(iii) Comprehensive platform for reproducing DRL algorithms. To facilitate reproducing SOTA DRL results, RLzoo provides a comprehensive DRL platform where users can access to a wide range of representative DRL algorithms and environments. These environments and algorithms include those that are relatively simple and thus suitable for learning and exploring DRL technologies. The platform also includes those that are advanced, such as RLbench and distributed DRL algorithms, which are useful for practical deployment; but not usually available in existing DRL libraries. Additionally, we augment the DRL platform with an *interactive training terminal*. This interactive terminal helps users construct and manage DRL agents, and tune hyper-parameters.

RLzoo is open-sourced on Github[1] in December, 2019. It has attracted numerous education and industry users, and become the key library that implements a wide range of demonstrations for a DRL textbook [19] published by Springer. In the following of this paper, we will describe the design of RLzoo, and compare RLzoo with existing DRL libraries in detail.

2 RLzoo Design

In this section, the design of RLzoo is introduced in detail. We start with an overview of the workflow, then describe its major APIs with a concrete code example.

[1] https://github.com/tensorlayer/RLzoo

```
1  from rlzoo.common.env_wrappers import build_env
2  from rlzoo.common.utils import call_default_params
3  ## Step 1: select and build the environment
4  env_type = 'classic_control'
5  env_name = 'Pendulum-v0'
6  env = build_env(env_name, env_type) # Build environment
7  ## Step 2: choose the algorithm and get default hyper-parameters
8  from rlzoo.algorithms import TD3 # Choose algorithm
9  alg_params, learn_params = call_default_params(env, env_type, 'TD3') # Create configuration
10 ## Step 3: create the RL agent and lauch learning process
11 agent = TD3(**alg_params) # Construct agent
12 agent.learn(env, 'train', **learn_params) # Launch training
```

Listing 1: Sample RLzoo program

2.1 Overview

The workflow of RLzoo is shown in Fig. 1. Four steps are necessary for training/testing: Step ❶. Select the environment and the RL algorithm; Step ❷. Pass in or call default hyper-parameters for the algorithm and learning process; Step ❸. Start the training or testing. An additional step ❹ for re-configuration and iterative learning may be applied when comparisons among different algorithms and settings are required.

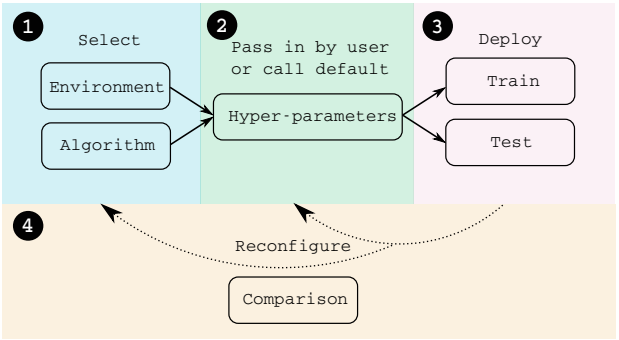

Figure 1: An overview of RLzoo usage with four key steps

2.2 Expressive and flexible APIs

RLzoo is a Python library built on top of TensorFlow [20] and TensorLayer [21]. When designing its Python APIs, we have the following goals in mind: (i) We want to provide *expressive* APIs so that DRL agents use the minimal amount of code; (ii) We want the APIs to remain *flexible* so that the DRL agents can be easily customised for different user applications.

We show the high-level APIs of RLzoo using Listing 1. To declare a DRL agent, RLzoo users need to choose and build an environment (i,e., `Pendulum-v0`) for this agent (line 4∼6). They then decide a DRL algorithm: `TD3` (line 8) to work with the chosen environment. To build the agent, the users first obtain its default construction parameters (line 9). The algorithm parameters (`alg_params`) are used for constructing a DRL agent (line 11). This agent then launches its training process (line 12) given the environment and hyper-parameters (`learn_params`). As we can see in this program, a DRL agent can be declared with 3 abstracted steps (9 lines of code). A summary of API functions and descriptions are provided in Table 1. The details of building the environments, importing the DRL algorithms, and constructing the DRL agents are hidden by the expressive API calls provided by RLzoo. In the following, we will discuss the details of these API calls.

Building learning environment. DRL environments are often imported from external libraries, e.g., OpenAI Gym. To hide the difference in the APIs of using these libraries, RLzoo provides an abstracted function: `build_env()` for importing environments. This function takes the environment name `env_name` and its type `env_type` (line 6 in

Function	Description
env = **build_env**(*EnvName*, *EnvType*)	Return the built environment instantiation with the name and type of it.
alg_params, *learn_params* = **call_default_params**(*env*, *EnvType*, *AlgName*)	Return two dictionaries of default hyper-parameters w.r.t. environments and algorithms.
agent = eval(*AlgName*+'(**alg_params*)') *agent*.**learn**(*env*, *mode*='train', *render*=False, ***learn_params*)	Instantiate the class of DRL agent. Launch training/testing process with the agent.

Table 1: RLzoo API

Listing 1). It automatically builds the environment by manipulating external libraries and transparently operates this environment within the DRL agent.

Obtaining default agent configuration. Initialising a DRL agent needs massive parameters (e.g., the parameters for instructing the DRL algorithms and those for controlling the training process). Letting RLzoo users decide all these parameters makes RLzoo difficult to adopt among general users. To address this, RLzoo provides default pre-tuned parameters for its DRL agents. These parameters are obtained through the `call_default_params()` function. This function returns the pre-tuned algorithm parameters in the dictionary `alg_params`, and the learning hyper-parameters in the dictionary `learn_params`.

Customising agents. RLzoo allows users to easily customise DRL agents. This is achieved through providing an intuitive manner for configuring the algorithm and learning hyper-parameter dictionaries. For example, to customise the neural networks used within the DRL algorithm, users can access the default neural networks through the key: `'net_list'`. They can replace the default networks with custom neural networks. These neural networks follow a shared `'Model'` interface. They can thus be seamlessly integrated within the agent environment. Following the same manner, RLzoo users can customise other algorithm parameters, e.g., optimisers, and learning hyper-parameters like learning rate and batch size.

Constructing and manipulating agents. RLzoo makes the construction and manipulation of DRL agents easy and efficient. All DRL methods in RLzoo can be instantiated to be an `agent` (line 11 in Listing 1). This allows these agents to be manipulated consistently. All these agents can share utility functions that are pre-implemented within the base agent class, such as the `learn()` function which launches the training process or evaluates the performance of the agents.

2.3 Adaptive agent construction

Using RLzoo APIs, users can implement a wide range of DRL agents. This however introduces challenges. These agents usually consist of external environments and custom agent's modules, while those environments and modules may not be compatible with the existing components in the RLzoo library (e.g., some environments might produce image observations while the other produce vectors).

Existing DRL libraries often rely on users to *manually* resolve this compatibility issue. Figure 2 contains a typical architecture of a DRL agent (ignoring the three adaptors added by RLzoo). In this agent, if a new environment is provided, users would have to manually update the neural networks (e.g., MLP and CNN) to process the new observations. This manual update needs to propagate through the agent, e.g., updating the policies, then the action types. Relying on users to manually make all these updates incurs high development costs. It can also make the expertise of DRL become a prerequisite for using DRL libraries.

RLzoo wants to automate the process of constructing a DRL agent. Our key idea is to embed numerous *adaptors* between agent components. These adaptors infer the type of the output from upstream agent components and dynamically compute the input for downstream components. The adaptors will be called in sequence and automatically re-configure the entire agent.

We show how a DRL agent is being automatically constructed in Figure 2. The *observation adaptor* is placed between the observation from the environment and the neural network (see ❶). This adaptor infers the type of observations

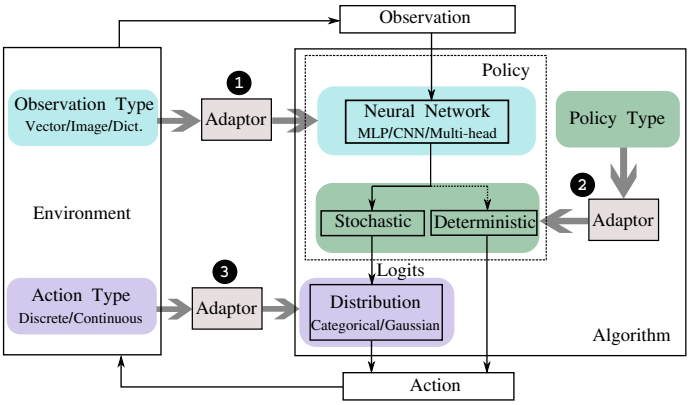

Figure 2: Adaptive agent construction process.

and produces different types of neural networks, e.g., a MLP for a vector, the CNN for an image, and a multi-head architecture for a hybrid dictionary. The *policy adaptor* is placed in between the algorithm selection and the policy output (see ❷). Based on the stochastic/deterministic nature of the DRL algorithm, this adaptor produces corresponding policy outputs for each selected DRL algorithm. The *action adaptor* is placed in between the policy (stochastic only) and the environment (see ❸): if the environment requires discrete action, this final adaptor will produce a categorical distribution to represent the action; if the action needs to be continuous, the adaptor produces policy output as a diagonal Gaussian distribution.

2.4 Comprehensive DRL platform

Making RLzoo useful for users introduces unique challenges. First, these users may not have the expertise of DRL to implement agents that are suitable for their applications. Second, training the agents to reach high accuracy often requires extra DRL knowledge in comprehending training results, and tuning hyper-parameters.

To address these challenges, we want to make RLzoo a comprehensive DRL platform. To avoid making DRL expertise a prerequisite, RLzoo provides a large collection of pre-defined DRL environments and algorithms which can be directly leveraged by users. RLzoo makes the tuning of DRL agents easy for non-experts of DRL. This is achieved through an interactive training terminal. This terminal provides useful functionalities for managing DRL agents and tuning their performance.

Pre-defined DRL algorithms and environments. Our choice for pre-defined DRL environments and algorithms is driven by the following observation: many users like to first verify the benefits of DRL technologies by starting with simple DRL algorithms and environments. They will gradually move to advanced DRL algorithms/environments once they realise the need for improving the performance of DRL agents.

RLzoo supports both simple and advanced training environments. Many simple environments have been integrated within the library, including Atari, Box2d, Classic control, MuJoCo, Robotics in OpenAI Gym, and DeepMind Control Suite. These environments cover most of the classical DRL benchmarks we are aware of. For advanced applications such as robot learning, RLzoo supports advanced environments in RLBench [12]. These environments can produce more realistic observations in dictionary/tuple type which can significantly improve the performance of DRL agents in practice.

RLzoo provides a large number of DRL algorithms. Classical DRL algorithms, including the Deep Q-Network [1] and its variants [22, 23, 24, 25] in the discrete action spaces, are pre-implemented in RLzoo. Many state-of-the-art DRL algorithms, which often achieve better agent performance, are pre-implemented as well. Examples include hindsight experience replay (HER) [26], deep deterministic policy gradient (DDPG) [2], twin delayed deep deterministic policy gradient (TD3) [27], soft actor-critic (SAC) [28], advantage actor-critic (A2C) [29], asynchronous advantage actor-critic (A3C) [29], proximal policy optimisation (PPO) [30], distributed proximal policy optimization (DPPO) [15], trust region policy optimisation (TRPO) [10].

Interactive training terminal. The interactive training terminal is based on the Jupyter Notebook. Using this terminal, RLzoo users can choose pre-defined algorithms, environments, and hyper-parameters intuitively. The notebook manages

the training of the DRL agents and automatically collects and analyses their training metrics. The notebook displays much useful information for agents including their configurations, learning status (e.g., training steps and instant rewards), and training results (e.g., averaged rewards over time and values of loss functions). Based on the information, RLzoo users can estimate the effects of changes made in the algorithms, environments, and hyper-parameters, thus improving their efficiency in tuning the performance of a DRL agent.

3 Evaluation

In this section, we compare RLzoo with other DRL libraries in terms of the supported algorithms, supported environments and their API designs. We choose the following popular libraries as baseline: OpenAI Baselines [13], Tianshou [16], Coach [14], ReAgent [31], garage [32], keras-rl [17], MushroomRL [33] and Tensorforce [18].

Library	# Algo.	# Env.	Image	Vector	Dict.	LoC
RLzoo	12	7	✓	✓	✓	4
Baselines	9	5	✓	✓	✓	N/A
Tianshou	8	5	✓	✓	✓	15-20
Coach	11	8	✓	✓	✗	N/A
ReAgent	4	3	✓	✓	✗	5
garage	9	6	✓	✓	✗	5-10
keras-rl	3	5	✓	✓	✓	10-15
MushroomRL	9	7	✓	✓	✗	5-10
Tensorforce	8	5	✓	✓	✓	5-15

Table 2: Comparison of different DRL libraries.

Algorithms. We first evaluate the algorithm support. All algorithms we considered in this comparison include DQN, HER, Rainbow [34], vanilla policy gradient (VPG) [9], A2C, A3C, actor-critic with experience replay (ACER) [35], actor critic using Kronecker-factored trust region (ACKTR) [36], DDPG, TD3, SAC, PPO, DPPO, TRPO, as well as the variants of DQN like double DQN [22], dueling DQN [23], Retrace [37], noisy DQN [24], distributed DQN [38], prioritized experience replay (PER) [25], quantile regression DQN (QR-DQN) [39], N-step Q-learning [29], normalized advantage functions (NAF) [40] and Rainbow [34]. As we can see from Table 2, RLzoo supports 12 DRL algorithms, whereas Coach supports 11 algorithms and other libraries support less than 10 algorithms. A key difference between RLzoo and other libraries is its support of *distributed* DRL algorithms, which makes RLzoo one of the few libraries that support distributed DRL algorithms such as DPPO. This type of algorithms is increasingly critical because practitioners have recently achieved great success of training DRL agents using parallel learning framework [15].

Environments. We then evaluate the environment support. The environments include: (1) Atari, Box2d, Classic control, MuJoCo, Robotics in OpenAI Gym (counted separately); (2) DeepMind Control Suite; (3) RLBench [12]; (4) Roboschool; (5) PyBullet [41]. As shown in Table 2, RLzoo supports 7 environments, making it among those libraries, e.g., Coach and MushroomRL, that provide a large collection of environments. A key feature for RLzoo is its support for all observations types (e.g., Vector, Image, and Dictionary). The Dictionary in this paper indicates either a *dictionary* or a *tuple* type in Python, which is literally a collection of sub-data with different shapes. The other library: keras-rl, which can offer the same full support, only provide 3 DRL algorithms, whereas RLzoo can support 12 DRL algorithms. This shows the importance of achieving adaptive agent construction in RLzoo: new observations can be automatically supported by all DRL algorithms. In addition, the full observation support also makes RLzoo the only library, as far as we know, that supports an important environment: RLbench. This environment has growing popularity due to the recent booming of robot learning applications. It produces complex observations that contain images, vectors, and dictionaries, making it difficult to be supported by existing libraries.

API expressiveness. We evaluate the API design by counting the lines of code (LoC) for declaring DRL agents. We exclude Baselines and Coach because they have only command-line interfaces. The LoCs here only consider necessary code for declaring agents, excluding other lines for importing libraries or assigning values for variables. As we can see in Table 2, RLzoo requires 4 LoCs to declare DRL agent while the ReAgent library comes as the second, costing 5 LoCs on average. Other programmable DRL libraries require users to write around 10 - 20 LoCs. In addition, RLzoo

differentiates with other libraries in terms of its support for customising agents. This makes RLzoo an attractive option for robot learning users who often need to (i) deal with RGB-D camera produced by the learning environment RLBench, and (ii) adopt customised network architectures like recurrent layers.

A complete comparison of RLzoo with other popular DRL libraries on (1) supported RL algorithms, (2) supported environments and (3) LoC are provided in Appendix A.

4 Conclusion

In this paper, we have described RLzoo, a novel DRL library that aims to help users develop, customise, and explore DRL technologies. We show that RLzoo can minimise the effort for developing DRL agents through its expressive and flexible API design. RLzoo enables users to efficiently manipulate external DRL environments and customise DRL algorithms, making use of an adaptive agent construction design. Users can further enjoy a comprehensive DRL platform where they can access to a large collection of pre-defined DRL environments and algorithms. RLzoo has become a popular DRL library on Github and attracted numerous users.

5 Acknowledgements

This work was supported by the funding for building AI super-computer prototype from Peng Cheng Laboratory (8201701524), the start-up research funds from Peking University (7100602564) and the Center on Frontiers of Computing Studies (CFCS) (7100602567).

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[3] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[4] Eugene Valassakis, Zihan Ding, and Edward Johns. Crossing the gap: A deep dive into zero-shot sim-to-real transfer for dynamics. *arXiv preprint arXiv:2008.06686*, 2020.

[5] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 5(2):1143–1150, 2020.

[6] Jongchan Park, Joon-Young Lee, Donggeun Yoo, and In So Kweon. Distort-and-recover: Color enhancement using deep reinforcement learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5928–5936, 2018.

[7] Ryosuke Furuta, Naoto Inoue, and Toshihiko Yamasaki. Fully convolutional network with multi-step reinforcement learning for image processing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3598–3605, 2019.

[8] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[9] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[10] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[12] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[13] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. URL `https://github.com/openai/baselines`, 2017.

[14] Itai Caspi, Gal Leibovich, Gal Novik, and Shadi Endrawis. Reinforcement learning coach, December 2017.

[15] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[16] Dong Yan Hang Su Jun Zhu Jiayi Weng, Minghao Zhang. Tianshou. URL `https://github.com/thu-ml/tianshou`, 2020.

[17] Matthias Plappert. keras-rl. URL `https://github.com/keras-rl/keras-rl`, 2016.

[18] Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Web page, 2017.

[19] Hao Dong, Zihan Ding, and Shanghang Zhang. *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Springer Nature, 2020.

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[21] Hao Dong, Akara Supratak, Luo Mai, Fangde Liu, Axel Oehmichen, Simiao Yu, and Yike Guo. Tensorlayer: a versatile library for efficient deep learning development. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1201–1204, 2017.

[22] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[23] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[24] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[25] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[26] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017.

[27] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[28] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[29] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Zhengxing Chen, Yuchen He, Zachary Kaden, Vivek Narayanan, and Xiaohui Ye. Horizon: Facebook's open source applied reinforcement learning platform. *arXiv preprint arXiv:1811.00260*, 2018.

[32] The garage contributors. Garage: A toolkit for reproducible reinforcement learning research. URL `https://github.com/rlworkgroup/garage`, 2019.

[33] Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Mushroomrl: Simplifying reinforcement learning research. URL `https://github.com/MushroomRL/mushroom-rl`, 2020.

[34] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[35] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[36] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288, 2017.

[37] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

[38] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

[39] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[40] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[41] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. URL `http://pybullet.org`, 2016–2019.

A Comparison Table

	RLzoo	Baselines	Tianshou	Coach	ReAgent	garage	keras-rl	MushroomRL	Tensorforce
DQN	✓	✓	✓	✓	✓	✓	✓	✓	✓
DQN variants	4	4	2	8	4	1	3	4	4
HER	✓	✓	✗	✓	✗	✓	✗	✗	✗
Rainbow	✗	✗	✗	✓	✗	✗	✗	✗	✗
VPG	✓	✗	✓	✓	✗	✓	✗	✓	✓
A2C	✓	✓	✓	✓	✗	✗	✗	✓	✓
A3C	✓	✗	✗	✓	✗	✗	✗	✗	✓
ACER	✗	✓	✗	✓	✗	✗	✗	✗	✗
ACKTR	✗	✓	✗	✗	✗	✗	✗	✗	✗
DDPG	✓	✓	✓	✓	✗	✓	✓	✓	✓
TD3	✓	✗	✓	✓	✓	✓	✗	✓	✗
SAC	✓	✗	✓	✓	✓	✓	✗	✓	✗
PPO	✓	✓	✓	✓	✗	✓	✗	✓	✓
DPPO	✓	✗	✗	✗	✗	✗	✗	✗	✗
TRPO	✓	✓	✗	✗	✗	✓	✗	✓	✓
# of Alg.	12	9	8	11	4	9	3	9	8
Atari	✓	✓	✓	✓	✓	✓	✓	✓	✓
Box2D	✓	✓	✓	✓	✓	✓	✓	✓	✓
Classic	✓	✓	✓	✓	✓	✓	✓	✓	✓
MuJoCo	✓	✓	✓	✓	✗	✓	✓	✓	✓
Robotics	✓	✓	✗	✓	✗	✓	✓	✓	✓
Control Suite	✓	✗	✗	✓	✗	✓	✗	✓	✗
RLbench	✓	✗	✗	✗	✗	✗	✗	✗	✗
Roboschool	✗	✗	✗	✓	✗	✗	✗	✗	✗
PyBullet	✗	✗	✓	✓	✗	✗	✗	✓	✗
# of Env.	7	5	5	8	3	6	5	7	5
LoC	4	N/A	15-20	N/A	5	5-10	10-15	5-10	5-15

Table 3: A complete comparison of RLzoo with other popular DRL libraries on: (1) supported RL algorithms, (2) supported environments, and (3) LoC. In "# of Alg.", all DQN variants are counted as one type.