RLCard: A Platform for Reinforcement Learning in Card Games

Daochen Zha[1*] , **Kwei-Herng Lai**[1*] , **Songyi Huang**[2*] , **Yuanpu Cao**[1] ,
Keerthana Reddy[1] , **Juan Vargas**[1] , **Alex Nguyen**[1] , **Ruzhe Wei**[1] , **Junyu Guo**[1] and **Xia Hu**[1]
[1]Department of Computer Science and Engineering, Texas A&M University, College Station, USA
[2]Simon Fraser University, BC, Canada
{daochen.zha, khlai037, keerthana96, juancvg, nguyen.alex, guojunyu, xiahu}@tamu.edu,
songyih@sfu.ca, yuanpucao@gmail.com, ruzhe.wei@outlook.com

Abstract

We present RLCard, a Python platform for reinforcement learning research and development in card games. RLCard supports various card environments and several baseline algorithms with unified easy-to-use interfaces, aiming at bridging reinforcement learning and imperfect information games. The platform provides flexible configurations of state representation, action encoding, and reward design. RLCard also supports visualizations for algorithm debugging. In this demo, we showcase two representative environments and their visualization. We conclude this demo with challenges and research opportunities brought by RLCard. A video is available on YouTube[1].

1 Introduction

Reinforcement learning (RL) is a promising paradigm towards Artificial Intelligence (AI). Through interacting with the environment, RL aims to train an agent to make sequential decisions to achieve a goal by exploiting reward signals [Sutton and Barto, 2018]. With deep neural networks as function approximators, deep reinforcement learning (DRL) has recently achieved breakthroughs in various domains, such as Atari games [Mnih *et al.*, 2015], Go game [Silver *et al.*, 2017], and continuous control [Lillicrap *et al.*, 2015].

Out of these achievements, however, DRL is still immature and not ready to be applied in many real-world problems. Current DRL algorithms are often less successful when dealing with long horizons, multiple agents, large decision space or sparse reward. Addressing these challenges is essential to bringing DRL to broader applications.

Card games are ideal testbeds for advancing DRL since many games have one or more of the above challenges. For example, Texas Hold'em is played by multiple players with large decision space, where each player needs to play against the other player and reason about the other players' cards that are hidden from her sight. Another example is a popular Chinese poker game Dou Dizhu, which suffers from long sequences, sparse reward, and large action space with an explosion of card combinations. These card games are nice abstractions of many real-world problems. Moreover, card games are easy to understand with huge popularity. We usually do not need to spend efforts on learning the rules before we can dive into algorithm development.

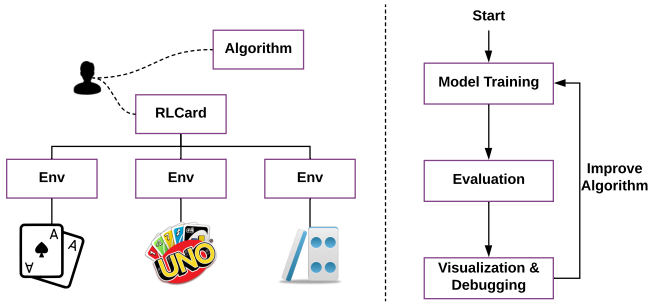

Figure 1: RLCard wraps card games with unified interfaces (**left**). RLCard helps users develop algorithms (**right**).

In this demo, we present RLCard[2], a platform designed for reinforcement learning research and development in card games. RLCard aims at providing easy-to-use interfaces and evaluation tools so that users can focus on algorithm development instead of engineering efforts on games.

2 RLCard Platform

An overview in shown in Figure 1. RLCard provides unified interfaces for seven popular card games, including Blackjack, Leduc Hold'em (a simplified Texas Hold'em game), Limit Texas Hold'em, No-Limit Texas Hold'em, UNO, Dou Dizhu and Mahjong. Moreover, RLCard supports flexible environment design with configurable state and action representations. Last but not least, RLCard provides visualization and debugging tools to help users understand their algorithms.

2.1 Interface Design

In RLCard, each agent has a local view of the game. The states are all the possible observations from the view of an agent, such as cards in hand and community cards. The actions are the legal moves of a player, such as "check" in Leduc Hold'em and "33" in Dou Dizhu.

[*]Those authors contributed equally to this project.
[1]https://youtu.be/krK2jmSdKZc
[2]https://github.com/datamllab/rlcard

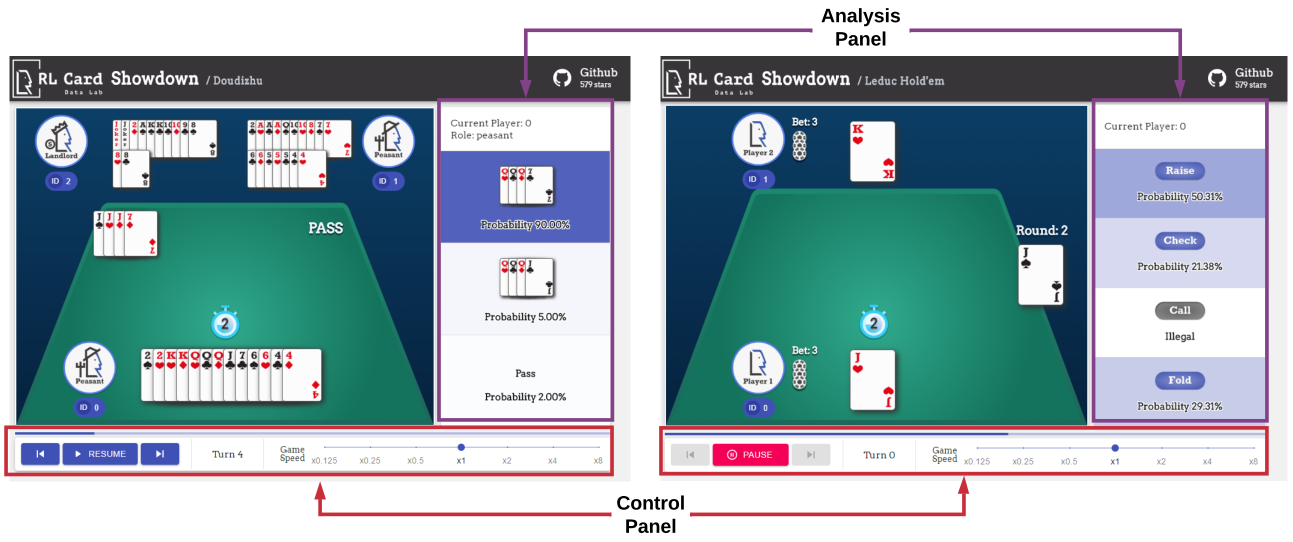

Figure 2: Visualization modules in RLCard of Dou Dizhu (**left**) and Leduc Hold'em (**right**) for algorithm debugging. The Control Panel provides functionalities to control the replay process, such as pausing, moving forward, moving backward and speed control. The Analysis Panel displays the top actions of the agents and the corresponding probabilities.

```
# Initialize the environment
env = rlcard.make('doudizhu')

# Initialize random agents
agent = RandomAgent()
env.set_agents([agent, agent, agent])

# Generate data from the environment
trajectories, payoffs = env.run()
```

Listing 1: Example function calls.

Basic Interface We provide a `run` function which conveniently collects the data of a complete game. It directly generates payoffs and game data, which are organized as transitions, i.e., (state, action, reward, next_state, done). This interface is designed for algorithms that do not need to traverse the game tree. An example of running Dou Dizhu with three random agents is sown in Listing 1.

The obtained trajectories can be used to train any deep reinforcement learning algorithms such as Deep Q-Learning (DQN) [Mnih *et al.*, 2015]. The payoffs provide the results of the game, such as win rates. One can also easily replaces `RandomAgent` with her own agent.

Advanced Interfaces We also provide interfaces that operate upon the game tree. We define a `step` function, which moves the environment to the next state given the current action. To enable traversing backward, we provide a `step_back` function, which traverses back to the previous state. The deign of `step` and `step_back` is similar to traditional tree-based interface. Specifically, `step` is corresponding to accessing child node, and `step_back` would access the parent node. This design enables flexible node visiting strategies of the game tree, such as external sampling MCCFR [Lanctot *et al.*, 2009].

2.2 Baseline Algorithms

We implement several baseline algorithms so that users can benchmark their algorithms against these existing models. We implement two deep reinforcement learning algorithms, including DQN [Mnih *et al.*, 2015], and Neural Fictitious Self-Play (NFSP) [Heinrich and Silver, 2016]. NFSP is a deep reinforcement algorithm adapted for imperfect information games. We also implement Counterfactual Regret Minimization (CFR) [Zinkevich *et al.*, 2008], which searches for the best strategies upon the game tree. In addition, we implement some rule-based agents as baselines. Many of the baseline results are presented in [Zha *et al.*, 2019a].

2.3 Evaluations and Visualizations

RLCard provides evaluation and visualization tools to help users understand their algorithms, as shown in Figure 2. Firstly, RLCard has a leader-board module, where users can easily compare a new algorithm with existing baselines, i.e., the pre-trained models and the rule baselines. Secondly, RLCard supports visualizations of replay data to enable algorithm debugging. One can analyze the top actions of the agents with the visualization modules to understand the strengths and weaknesses of the agents. The users can also put their trained models into RLCard as baselines.

3 Conclusions

Card games are nice abstractions of many real-world problems. RLCard brings many challenges, such as long horizons, multiple agents, large decision space and sparse reward, and also research opportunities for tackling these challenges. In the future, we plan to incorporate more recent RL techniques, such as PPO [Schulman *et al.*, 2017], SAC [Haarnoja *et al.*, 2018], and better replay buffer [Zha *et al.*, 2019b]. We will also support more card games, such as Gin Rummy.

Acknowledgements

We would like to thank JJ World Network Technology Co., LTD for the generous support, Chieh-An Tsai[3] for user interface design, and Lei Pan[4] for the help in visualizations.

References

[Haarnoja *et al.*, 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[Heinrich and Silver, 2016] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

[Lanctot *et al.*, 2009] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. In *NeuIPS*, 2009.

[Lillicrap *et al.*, 2015] 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.

[Mnih *et al.*, 2015] 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, 2015.

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

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Zha *et al.*, 2019a] Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo, and Xia Hu. Rlcard: A toolkit for reinforcement learning in card games. *arXiv preprint arXiv:1910.04376*, 2019.

[Zha *et al.*, 2019b] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and X. X. Hu. Experience replay optimization. In *IJCAI*, 2019.

[Zinkevich *et al.*, 2008] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *NeuIPS*, 2008.

[3]https://anntsai.myportfolio.com/
[4]lpa25@sfu.ca