Rank TD: End-to-End Robotic Reinforcement Learning without Reward Engineering and Demonstrations

○ ShengKai Huang, Student Member, University of Electro-Communications, huang@taka.is.uec.ac.jp
Masaru Takizawa, University of Electro-Communications
Shunsuke Kudoh, Member, University of Electro-Communications
Takashi Suehiro, University of Electro-Communications

Abstract: The development of reinforcement learning and deep neural networks allow us to train a decision-making system for robots by the end-to-end method, which directly leverages raw sensory inputs, and outputs an action. Designing a reward function that not only reflects the goal of the task but also facilitates the agent's exploration, however, is tedious and challenging. This paper introduces a technique that allows agents to explore following the expert-designed state trajectory and take a balance between the creativity of agents and the rigid rules of the game shaped by prior knowledge. We investigate and evaluate our approach on a simple case and a complex robotic arm grasping-task. The results show that our method has a good application prospect in the sim2real field.

Key Words: Reinforcement learning, Manipulation, Pick and Place

1 Introduction

Reinforcement learning (RL) has recently achieved widespread success in sequential decision-making problems. Moreover, the function approximation approach based on deep neural networks (DNN) allows RL to train a policy for robots that directly reads raw sensory inputs, such as camera images. Designing a reward function that not only reflects the task goal but is also carefully shaped to efficiently explore, however, is still a common challenge.

For example, Popov et al.[1] used a reward function that consists of five complicated terms carefully weighed to get a stacking policy. It limits the applicability of RL because it takes much time to shape an appropriate reward function. On the other hand, oversimplified reward functions such as a binary signal could cause serious sparse reward issues. Although there are some data augmentation methods such as Hindsight Experience Replay[2] that could solve the partial sparse reward issue, it, however, is not easy to apply in the high-dimension state, such as images.

The other tricky issue is that current image-policy-based robot learning depends on demonstrations, which is a frequently-used method of Imitation Learning (IL). Demonstrations that are generated by a human to initialize policy or make a replica of the demonstration policy include DDPGfD[3] and behavioral cloning (BC). BC or IL approaches, however, are limited because they do not consider the task or environment. Furthermore, human demonstrations can be suboptimal because humans also make mistakes or do some redundant actions. Moreover, agents will get into trouble when they encounter states that do not belong to the demonstration set.

An important reason for the combination of RL and IL is that IL significantly reduces the exploration space of agents by introducing prior knowledge. IL, however, also over-define the rules of the game and limit the creativity of the agent. This paper aims to find a balance between the creativity of agents and the rigid rules of the game shaped by prior knowledge.

In this paper, we introduce a technique called Rank Temporal-Difference (Rank TD), which allows agents to explore by following the expert-designed state trajectory. Furthermore, it can combine with any model-free RL algorithm. We build its mathematical model and discuss the property by analyzing a simple case. Then, we prove the correctness of the property. Finally, we train an agent with an image-based policy on pick-and-place task simulation. We employed domain randomization in simulation to guarantee the potential of policy transfer from simulation to the real world without further training.

2 Mathematical Model

2.1 Preliminary

We follow the classic RL setting that can be represented as a Markov Decision Process (MDP) defined as a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mathcal{O})$, where $\mathcal{S}$ is the set of full states of the environment, $\mathcal{A}$ is the set of actions, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, γ is a discount factor, and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a deterministic state transition function. The decision process is partially observable and the agent receives observations o from the set of observations $\mathcal{O}$, followed by a sample process $o \sim \mathcal{G}(s)$.

2.2 Rank Temporal Difference

We first provide the general theoretical definition of a rank-based optimized problem, then provide three definitions and a critical theory.

Unlike the definition of optimal policy π^* in classic RL, in this paper, the optimal policy π^* in a goal-oriented reinforcement learning task is:

$$\pi^* \doteq \arg\max_{\pi} \mathtt{Pr}_{\tau\sim\pi}\left(s_T = \mathtt{goal}\right) \tag{1}$$

where trajectory $\tau\left(s_0, o_0, a_0, s_1, o_1, a_1 \cdots s_t, o_t, a_t \cdots s_T\right)$ is generated under the policy $\pi\left(a_t \mid o_t\right)$.

Rank function is defined as a mapping $\mathtt{rank} : \mathcal{S} \to \mathbb{N}$, satisfied that if trajectory τ is generated under the optimal policy π^*, then

$$\mathtt{rank}\left(s_0\right) \leqslant \mathtt{rank}\left(s_1\right) \leqslant \cdots \leqslant \mathtt{rank}\left(s_t\right) \leqslant \cdots \leqslant \mathtt{rank}\left(s_T\right)$$

And the reward defined as

$$r\left(s_t, s_{t+1}\right) \doteq \begin{cases} -1, & \boldsymbol{s_{t+1} = terminal} \\ \mathtt{rank}\left(s_{t+1}\right) - \mathtt{rank}\left(s_t\right), & \boldsymbol{other} \end{cases}$$

The goal of optimization is to find an optimization policy $\pi^*\left(a_t \mid o_t\right)$.

$$\begin{aligned} \max_{\pi} \quad & \mathtt{Pr}_{\tau\sim\pi}\left(s_T = \mathtt{goal}\right) \\ s.t. \quad & cout(\mathtt{rank}\left(s_t\right) > \mathtt{rank}\left(s_{t+1}\right)) \leqslant \varepsilon, \quad \varepsilon \in \mathbb{N} \ . \end{aligned}$$

ε is a hyper-parameter that restricts the times of inverse rank transform during roll-out. When an episode arises more than ε inverse rank transforms, the episode would be terminated.

Table 1 Dynamic Transform Table

s_{t+1} \ a_t / s_t	⇐⇐	⇐	◯	⇒	⇒⇒
0	0	0	0	1	2
1	0	0	1	2	3
2	0	1	2	3	4
3	-1	-1	-1	4	5
4	2	3	4	5	6
5	3	4	5	6	-1
6	4	5	6	7	8
7	5	6	7	8	9
8	6	7	-1	-1	-1
9	7	8	9	10	11
10	8	9	-1	-1	-1
11	9	10	11	12	13
12	10	11	12	13	14
13	11	12	-1	-1	-1
14	12	13	14	15	16
15	13	14	15	16	17
16	-1	-1	-1	17	18
17	15	16	17	18	19
18	16	-1	-1	-1	-1
19	17	18	19	20	20

3 Experiment

3.1 Simple Tasks

Fig.1 Simple case: agent exists in a world with 21 discrete states spaces. The agent can choose one from action space $\mathcal{A} = \{\Leftarrow\Leftarrow, \Leftarrow, \bigcirc, \Rightarrow, \Rightarrow\Rightarrow\}$. The goal is to arrive in state 20.

First, we design a simple task (Fig.1). Assuming an agent exists in a world with state space $\mathcal{S} = \{0, 1, ..., 20\}$ and action space $\mathcal{A} = \{\Leftarrow\Leftarrow, \Leftarrow, \bigcirc, \Rightarrow, \Rightarrow\Rightarrow\}$ which means `two steps backward, one step backward, stand still, one step forward, two steps forward.` The agent initialized in 0 state each time, which $p\left(s_t = 0 | t = 0\right) = 1$. The deterministic environment dynamic transform $s_{t+1} \leftarrow \mathcal{P}\left(s_t, a_t\right)$ defined as Table 1. Red "-1" means that the agent falls into a trap and terminates the episode. Green "20" is the goal state for the agent. In this simple case, observation o_t is equivalent to the internal full state s_t, which $o_t = s_t$.

We designed two kinds of reward functions. The first is that the agent gets a bonus of 1 only when it reaches the target state, or a penalty -1 when it falls into the trap, and 0 at other times (Equation 2).

$$r_{\boldsymbol{sparse}}\left(s_t, s_{t+1}\right) \doteq \begin{cases} -1, & \boldsymbol{s_{t+1} = -1} \\ 0, & \boldsymbol{other} \\ 1, & \boldsymbol{s_{t+1} = 20} \end{cases} \tag{2}$$

The other reward is set as the temporal difference between the previous state and current state, and -1 when it falls into the trap (Equation 3).

$$r_{\boldsymbol{TD}}\left(s_t, s_{t+1}\right) \doteq \begin{cases} -1, & \boldsymbol{s_{t+1} = -1} \\ s_{t+1} - s_t, & \boldsymbol{other} \end{cases} \tag{3}$$

We try to find a policy $\pi\left(a_t \mid o_t\right)$ by the Proximal Policy Optimization (PPO) Algorithm [4] under two reward settings with same hyper-parameter, architecture, and random seed.

The result (Fig.2) shows that if the positive feedback signal is too sparse, the agent would learn nothing. On the contrary,

Fig.2 PPO cannot solve this task in a sparse reward setting; however, gets great results in the reward setting, which is based on temporal differences.

the immediate feedback signal (state temporal difference) setting, which could reflect the change of completing the task, allows learning to be more smoothly.

We can imagine that the rank function is a sequence of waypoints, which also represents the completeness of a task, as defined by the expert according to the expert's understanding of the task.

Secondly, we modify the simple case, drawing a useful conclusion.

Bottleneck Issue: Make sure the state transitions between the adjacent ranks are not too complicated. When it is difficult for an agent to explore the path from one rank to a higher rank, in other words, when the path becomes narrower, learning will be blocked. We call it *bottleneck issue*. For example, if we change rows 2, 3, and 4 of the original state transform table, such as Table 2.

Table 2 Narrower Version Dynamic Transform

s_{t+1} \ a_t / s_t	⇐⇐	⇐	◯	⇒	⇒⇒
2	0	1	2	3	-1
3	-1	-1	-1	4	-1
4	-1	-1	-1	5	-1

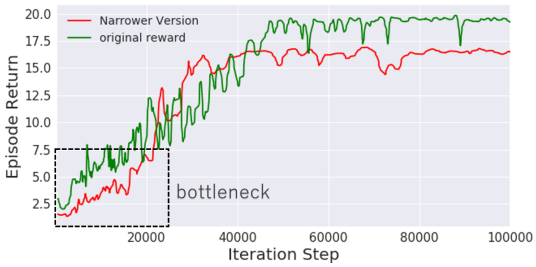


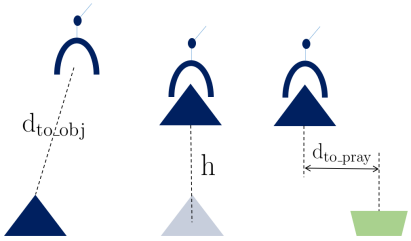
Fig.3 More complex state transform of the task makes it more difficult for the agent to explore. When applied in complex problems, if we fail to grasp the scale or granularity of the rank function, agents are unable to learn the good policy.

The result (Fig.3) shows that as the process of state transform becomes more complex and the ascending path becomes narrower, the difficulty of agent exploration increases.

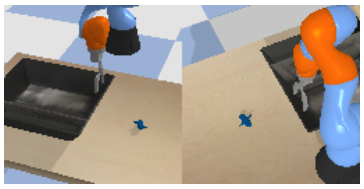
3.2 Pick and Place Task

finally, we design a *Pick and Place* robot task, where the robot-arm grasps an object and puts it in a tray.

The robot environment developed and published by Google Brain[6] is built on the PyBullet simulator[7].



(a) internal state $\mathcal{S}$



(b) observe image $\mathcal{O}$

Fig.4 Internal state and Observation

Table 3 rank function

rank	Internal Full State
0	$d_{to\_obj} > 57cm$
1	$37cm < d_{to\_obj} \leqslant 57cm$
2	$27cm < d_{to\_obj} \leqslant 37cm$
3	$18cm < d_{to\_obj} \leqslant 27cm$
4	$14cm < d_{to\_obj} \leqslant 18cm$
5	$9cm < d_{to\_obj} \leqslant 14cm$
6	$5cm < d_{to\_obj} \leqslant 9cm$
7	gripper not open
8	gripper open
9	$h \leqslant 1cm$
10	$1cm < h \leqslant 4cm$
11	$4cm < h \leqslant 7cm$
12	$7cm < h \leqslant 10cm$
13	$10cm < h \leqslant 15cm$
14	$d_{to\_pray} > 57cm$
15	$50cm < d_{to\_pray} \leqslant 57cm$
16	$40cm < d_{to\_pray} \leqslant 50cm$
17	$30cm < d_{to\_pray} \leqslant 40cm$
18	$23cm < d_{to\_pray} \leqslant 30cm$
19	$d_{to\_pray} \leqslant 23cm$
20	object drop into pray

There are many diverse and highly varied situations in robotic grasping. We prepared 90 various objects for the training phase, and 10 objects for the testing phase. We want to know whether agents can generalize various grasping strategies for different kinds of objects.

Training an end-to-end model in the real world requires too many samples. The current trend, therefore, is to learn an image-based policy in simulation and then transfer them to the real world, which is called *sim2real*.

The internal full state s_t is hard to measure or achieve in the real world but easy in the simulator. Thus, we could leverage the expert's prior knowledge to define a sequence of waypoints by the simulator's internal full states, and then train an image-based policy, which could also work in the real world.

For overcoming the *sim-real gap*, which is caused by a different sample process $o^{\text{sim}} \sim \mathcal{G}_{sim}(s)$ and $o^{\text{real}} \sim \mathcal{G}_{real}(s)$, we employed domain randomization[5] to facilitate a smooth domain transfer of the learned policy.

3.2.1 Task Description

Action $\mathcal{A}$: The robot-arm is 7DOF kuka-iiwa because this task does not need to change the orientation of the end-effector, thus the orientation is fixed vertically. The robot is controlled by 5-dimensional action. The first three dimensions are the relative change of cartesian coordinates$(\Delta x, \Delta y, \Delta z)$ of the end-effector, while the fourth dimension is the relative change of gripper's rotation angle$(\Delta\theta)$, and the last dimension is the instruction of the gripper open and closed (negative for closing and positive for opening).

Internal Full State $\mathcal{S}$: The Internal full states of the simulator as in Fig.4(a), include 1) the distance between the gripper and object d_{to_obj}; 2) the height of the object to be lifted h; 3) the distance between the gripper and pray d_{to_pray}; 4) gripper-pray and gripper-object collision detection.

Observation $\mathcal{O}$: The agent gets RGB observation with dimensions $256 \times 128 \times 3$ as in Fig.4(b), captured by two cameras that the one is located at the top-left of the robot, and the other one is located at the shoulder of robot. The image obtained at this location contains the relative positions of the pray, object, and gripper. Furthermore, we allow the GPU to render the shadow, which can effectively avoid the judgment error of the relative position caused by the angle of view.

Initial State Distribution : The gripper's initial position and the objects' scattered location are randomly generated in the legal region. The location of the pray is fixed and immovable.

Success Condition (goal) : The object falls into the pray.

Terminal Conditions : 1) The end-effector is outside the legal area; 2) The gripper collided with the pray; 3) Exceed the threshold ε of inverse rank transform times, which $cout(\texttt{rank}(s_t) > \texttt{rank}(s_{t+1})) > \varepsilon$; 4) Exceeds the maximum episode length (128).

3.2.2 Implemention

The rank function defined is based on the expert's understanding of the task using the internal state of the simulator as in Fig.4(a).

The policy is parameterized as a DNN φ. We adopt the PPO algorithm to find the optimal policy$\pi_\varphi \rightarrow \pi^*$. The policy net is shown as (Fig.5(b)). The padding way of CNN is all *valid*. CNN extracts the features from high-dimensional images, and the full connection layer interprets and uses the CNN features. After the tanh active function, the mean, and the standard deviation of the gaussian distribution were output by the full connecting layer, and action a was sampled from the gaussian distribution.

Input images are scaled in $[0, 1]$ for stable training, and the last layer's weight of actor net initialized with 0 before training, which means the mean and standard deviation initialized with 0 before training.

During one training epoch, 5 Kuka workers roll-out episodes data that is based on current policy π_{φ_t}, and evaluate the policy, which updates the value function the same as the classic PPO algorithm. Then, they update the policy to $\pi_{\varphi_{t+1}}$ based on the Trust Region policy optimization theory[8].

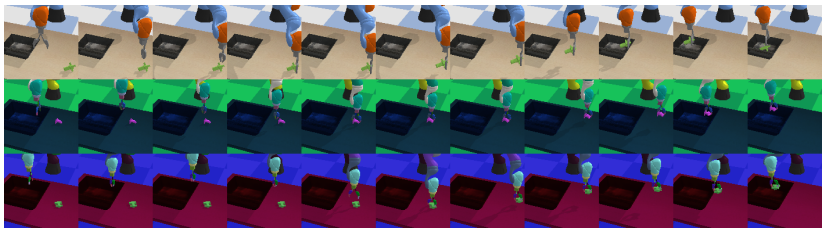
The discount γ in our experiment is 0.9; The inverse rank transform threshold ε is 3; max episode length is 128.

We experiment on a platform with Intel i7-8700k, 32Gb RAM, GTX 1080Ti, and cost 72 hours for 3 000 000 times policy iteration.

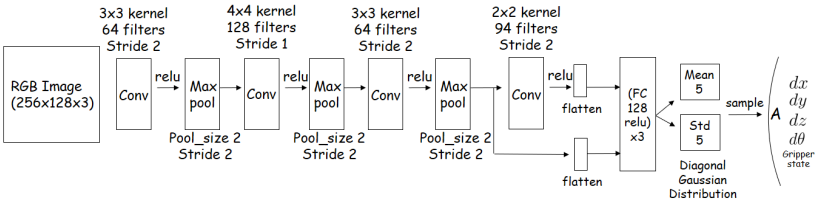
3.2.3 Result

We evaluate our method that the agent can act following the expert expected manner.

The learning curve is shown in the upper of Fig.3.2.3. An interesting phenomenon is that because there is a penalty for



(a) The first line is the unrandomized observation sequence. The following two lines are the sequence of observation with randomization.



(b) Actor Net Architecture. The agent learns an actor model in the simulator that only depends on the observe image. We concatenate the last two layers of features for better using global features. Action a is sampled from a multivariate gaussian distribution, the mean μ, and the diagonal standard deviation σ only depending on the input image.

Fig.5 Vision-Based Robotic Grasping

hitting the pray, the agent knows to avoid hitting the pray before grasping the object.

We evaluate our model on the test scenario. The object is never encountered during training. The success rate for the complete process was 26.4% (264/1000).

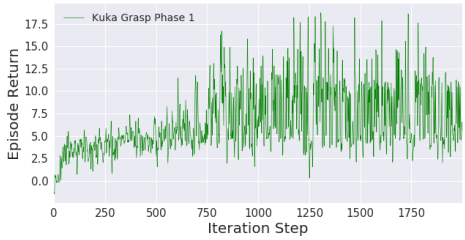


Fig.6 The upper is the training curve of phase 1, and the lower is the pre-grasp behavior, which avoids hitting the pray.

4 Conclusion

In this paper, we proposed an approach to RL learning without tedious reward engineering and human demonstrations. In the simple case, we verified the validity of the method and discussed the relationship between the partition granularity of rank function and the bottleneck issue. Then, We apply this method in a complex pick-and-place task, in which the decision-making only require single image.

There are still some parts could be improved. One is the representation of ovservation. We can deduct high-dimensional observation to low-dimensional state vectors by some unsupervised approach such as the AutoEncoder. Another is reducing the length of rank functions. It is difficult to construct the rank function of a complex task using single internal state variables. And if the rank is defined too long, it is hard to explore a high-rank state. One way to relieve this problem is to modify initial state distribution $p(s_0)$, proposed in DeepMimic[9], which makes high-rank state could aslo be fully explored.

References

[1] Popov I, Heess N, Lillicrap T, et al. Data-efficient deep reinforcement learning for dexterous manipulation[J]. arXiv preprint arXiv:1704.03073, 2017.

[2] Andrychowicz M, Wolski F, Ray A, et al. Hindsight experience replay[C]//Advances in neural information processing systems. 2017: 5048-5058.

[3] Vecerik M, Hester T, Scholz J, et al. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards[J]. arXiv preprint arXiv:1707.08817, 2017.

[4] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

[5] Tobin J, Fong R, Ray A, et al. Domain randomization for transferring deep neural networks from simulation to the real world[C]//2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2017: 23-30.

[6] Quillen, Deirdre Jang, EricNachum, OfirFinn, ChelseaIbarz, Julian Levine, Sergey.(2018).Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods.

[7] E. Coumans and Y. Bai. pybullet, a python module for physics simulation, games, robotics and machine learning. http://pybullet.org/, 2016–2017.

[8] Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization[C]//International conference on machine learning. 2015: 1889-1897.

[9] Peng X B, Abbeel P, Levine S, et al. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills[J]. ACM Transactions on Graphics (TOG), 2018, 37(4): 1-14.