Learning to Assign Credit in Reinforcement Learning by Incorporating Abstract Relations

Dong Yan, Shiyu Huang, Hang Su, Jun Zhu
Department of Computer Science and Technology, THU Lab for Brain and AI,
THBI Lab, Tsinghua University, Beijing, 100084, China

Abstract

Credit assignment is one of the most critical problems in reinforcement learning to discover which actions are responsible for rewards. It becomes more serious as reinforcement learning is applied to real-world scenarios where the decision process may involve thousands of actions. In this paper, we propose a novel framework that utilizes a computation process to assign credits for hundreds of thousands of non-terminal state-action pairs, in order to accelerate the learning speed. Specifically, we first abstract the states and actions of the original problem into a compact representation, which reduces the problem to a tractable size. Then, we solve the abstracted problem to obtain the optimal value function, which is the expected returns of future rewards. Finally, we use the derived value function to assign credits for state-action pairs of the original problem. We conduct extensive experiments on Doom, a complex 3D video game in which the reward signal is sparse. The experiment results demonstrate that our agent outperforms previous state-of-the-art agents in terms of both kill count and death number with a large margin. The effectiveness also manifests in an online competition of Doom, in which we achieved the 2nd place in the final.

1 Introduction

Credit assignment is one of the most fundamental challenges in reinforcement learning, which refers to how to assign the credit for the outcome of the multitude of decisions. In general, it is nontrivial since the ultimate success (or failure) is associated with a vast number of internal decisions in the course of play. Most iterative reinforcement learning algorithms simply backpropagate the influence of the final reward along the planning horizon step by step. Since the influence of a reward gets more and more diluted over time, they only work well if the horizon is short. However, most of the time, the length of the planning horizon is up to hundreds or even thousands, e.g., playing chess or robot navigation. Therefore, credit assignment becomes the vital point of reinforcement learning in real-world scenarios.

Recently, deep reinforcement learning (DRL) (Sutton and Barto, 2017) has achieved significant successes in various applications, including Atari (Mnih et al., 2013) and the game of Go (Silver et al., 2017). However, the issue of credit assignment becomes more serious for DRL in these complex tasks. For example, Montezuma's Revenge (MR) is an Atari game, in which the agent needs to collect keys to open the door. However, it is generally not feasible to implement an effective credit assignment since one should assign credits to all the images including keys. In this case, DRL usually fails on the task of Montezuma's Revenge even it can surpass human performance on some other Atari games.

Reward shaping (Ng, Harada, and Russell, 1999) solves the credit assignment problem by redesigning the reward function based on expert knowledge. By incorporating the expert knowledge, reward shaping is able to improve the agent performance without additional computation power. The main difficulty of reward shaping is that the expert knowledge is presented as concepts and principles, while the reward function is presented in numerical values. For the complex and dynamic environment, it is difficult if not impossible to transfer the expert knowledge into numerical forms correctly, even with large and continuous human efforts. OpenAI FIVE (OpenAI, 2018b) adopts massive computation resources and reward shaping to play DOTA2, a multi-player online game. It was trained with both huge amounts of computation power (180 game years per day) and considerable human efforts on reward shaping (an expert designed reward system which assigns credits to different actions and varies as the performance increases). However, the agent is still being easily defeated by professional players (OpenAI, 2018a).

To address the aforementioned issues, we introduce a computation based credit assignment framework. Our framework compresses the original problem into an abstracted form, and then solves it to obtain the value function as the guidance for credit assignment of the original problem. Instead of directly assigning credits in the original state-action space, our framework only requires to assign credits to several selected state-action pairs in the abstracted state-action space, and then propagates the credits to the whole space. Our method is able to **assignment credits for hundreds of thousands of state-action pairs in an automated manner**. It has boosted the performance of the vanilla DRL algorithm by providing a more rich, accurate, and fine-grained credit assignment than the handcrafted reward shaping.

Specifically, our framework consists of three stages: ab-

straction, planning and feedback. In the abstraction stage, we adopt first-order-logic (FOL), a widely used formal system, to abstract each state in DRL (original state) into a set of first-order-logic propositions (relational state). Note that multiple original states can be mapped to a same relational state, thus the number of relational states is much smaller than the number of original states. In the planning stage, we run the value iteration algorithm to obtain the optimal value function for each relational state in a feasible time without any approximation. In the feedback stage, the optimal value function for relation state is not the optimal value function for the original state, because multiple original states can be mapped to the same relation state. Therefore, we adopt the optimal value function as the potential function (Ng, Harada, and Russell, 1999), which is used in reward shaping to boost the performance of DRL algorithm running on original states.

We conduct experiments on a First-Person-Shooter (FPS) game Doom (Kempka et al., 2016), which is widely used as the test-bed for DRL algorithms. The original reward (killing the enemy) in Doom is pretty sparse and delayed, as the agent needs to explore in a complex map to find the enemy, then battle with the enemy till win. This procedure may involve more than hundreds of actions. Thus, handcrafted credit assignment on Doom has a performance upper bound due to the complexity of the game.

The evaluation involves three different settings, ablation study, known opponents comparison and real world competition, to obtain a comprehensive analysis of our framework. Firstly, for ablation study, we divide the derived credit assignment into several categories, each of which is used to train an individual agent. Then we analyze the effect of different credit assignment by testing the performance of those agents. Secondly, we compare our agent with previous state-of-the-art agents (Dosovitskiy and Koltun, 2016; Wu and Tian, 2017). Benefiting from the automated credit assignment, the experiment results show that our agent outperforms the others on both kill count and death number. Finally, we have attended a competition of Doom, in which we fought against unknown opponents on unknown maps to obtain a real-world test of our framework. We achieved the 2nd place in the final, and killed the same number compared with the champion but with 2 more suicides.

2 Related Work

Deep reinforcement learning and relational reinforcement learning are the most related works to our framework. We review their recent progress in this section.

2.1 Deep Reinforcement Learning

Recent achievements of reinforcement learning (RL) are partially due to the combination of deep neural networks (Mnih et al., 2015), with breakthroughs in both novel architectures (Mnih et al., 2016) and applications (Silver et al., 2016). One profound example is applying deep RL to games (Silver et al., 2016; Mnih et al., 2013), which serve as effective test beds to explore the techniques. In this scenario, an agent interacts with its environment to learn a policy, i.e., to decide the appropriate actions in order to reach a desirable state. Representative works on more challenging tasks of 3D games (Kempka et al., 2016) include Deep Recurrent Q-learning (Lample and Chaplot, 2017) and successor representations (Kulkarni et al., 2016). Besides, curriculum learning is demonstrated to be effective for training an agent step by step (Wu and Tian, 2017). Although deep RL methods have shown remarkable results, there still exist numerous challenges that are not well addressed, among which the exploration-exploitation dilemma is a critical one. The average branching factor of decision spaces hovers around 30 for Chess and 300 for Go, but even a video game such as Doom or StarCraft has an order of magnitudes larger branching factor. It is imperative to develop techniques that can guide the explorations effectively.

2.2 Relational Reinforcement Learning

Incorporating the power of logic with learning methods has been studied for decades. Relational reinforcement learning (Džeroski, De Raedt, and Driessens, 2001) presents a learning technique that combines reinforcement learning with relational learning. Due to the use of a more expressive representation language to represent states, actions and Q-functions, it is effective for tasks with structural internal representations. With the rise of deep learning, recent approaches combing relational representations with deep neural networks. Deep relational reinforcement learning (Zambaldi et al., 2018) uses self-attention to iteratively reason about the relations between entities and guide a model-free policy. In the following section, we adopt techniques from the relational reinforcement learning to build the compact representation of the original problem and solve it.

3 Method

In this section, we first give an overview of our framework, and then present the details of incorporating the first-order-logic within a computation process to address the credit assignment problem in deep reinforcement learning step by step.

3.1 Overview

The Markov decision process (MDP) provides a mathematical framework for modeling decision making, and we use it to describe our framework formally. An MDP is a 5-tuple, $M =< S, A, T, R, \lambda >$, where each element in the tuple indicates for state, action, transition matrix, reward function and the discount factor, correspondingly. Without losing generality, we take the Deep Q Network (DQN) (Mnih et al., 2013) as a representative of DRL algorithms, which utilizes a convolution neural network to learn the Q function by sampling from the environment. In the language of MDPs, the update equation of DQN is :

$$Q(s,a) \leftarrow Q(s,a)+\alpha[R(s,a,s')+\lambda \max_{a'} Q(s',a')-Q(s,a)], \tag{1}$$

where $Q(s,a)$ is the Q value of each state-action pair, $R(s,a,s')$ is the reward function and α is the learning rate.

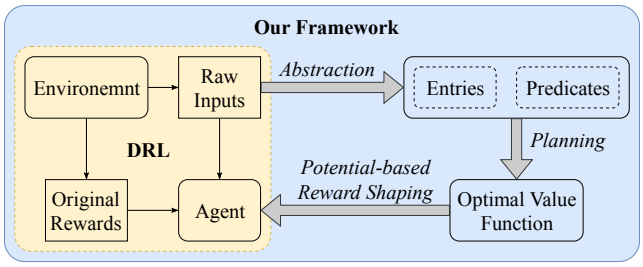

Figure 1: The working procedure of our mechanism. Generate the credits for state-action pairs of the deep reinforcement learning in three stages, abstraction, planning and potential-based reward shaping.

Without credit assignment, in many real-world scenarios, the reward $R(s, a, s')$ only appears at the end of a trajectory, e.g., playing chess or finding out the exit of a maze. In that case, the learning signal needs to propagate through all state-action pairs one by one to update their Q value, which can slow down the converging speed of DRL dramatically.

We introduce a Relational Markov Decision Process (RMDP) to generate additional rewards since the design of reward function is the key point of credit assignment. The RMDP, $M_r =< S_r, A_r, T_r, R_r, \lambda_r >$, is a special type of MDP, where each state is a set of FOL propositions. The main advantage of RMDP is that it is easy to incorporate expert knowledge to design the reward function while supporting all those MDP solving algorithms.

We can therefore incorporate RMDP to assign credits in an MDP as

$$R(s, a, s') = R(s, a, s') + \lambda V_r(f_s(s')) - V_r(f_s(s)), \quad (2)$$

where V_r is the optimal value function for state in RMDP, and f is an abstraction function that maps states S in MDP to relational states S_r. We solve the RMDP to obtain the optimal value function of each state, and use it to assign credits for the DRL.

Fig. 1 illustrates the working flow of our framework, which consists of three steps, abstraction, planning and feedback through reward shaping. First, the abstraction stage converts the raw inputs into entries and predicates as the state of the RMDP. Then, the planning stage solves the RMDP to generate the optimal value function. Finally, we use it to reshape the original reward to accelerate the training process of the agent. The details will be introduced in the following sections as in the same order.

3.2 Abstraction

The abstraction stage maps multiple states in the MDP into a single state in an RMDP. Generally, states in MDP are raw images or numerical vectors, e.g., the screen of Atari game or the game board of Go, while in RMDP states are represented by first-order-logic propositions. Thus, the input of abstraction stage is original states and actions of MDP and the output is first-order-logic propositions. This can be formally put as :

$$S_r = f_s(S),\ A_r = f_a(A), \quad (3)$$

where the functions f_s and f_a map the state S and action A in MDP to state S_r and action A_r in RMDP, respectively. S and A are raw states and actions of the original task, while the corresponding S_r and A_r are expressed in the form of FOL propositions.

The state of RMDP only reserves important features such that the states can be significantly compressed. For instance, we can keep the foreground information in the state vector of S in the vision-related tasks while omitting the background information. Specifically, all states in the MDP which can be represented by the same set of FOL propositions will be mapped to the same state in RMDP. Thus, different states with the same object but different backgrounds in MDP will be mapped to the same state in RMDP, which facilities the credit assignment by greatly reducing the state dimensions.

The implementation of abstraction stage depends on the input form of the original task and which type of information is decided to be retained. Various kinds of techniques can be used for abstraction. For simple grid games, a rule-based transformation is enough, while for vision-related tasks, we may use object detection, image segmentation, and localization. For instance, assume the input image is an apple on a box. At first, we adopt an object detection neural network to generate two bounding boxes for the apple and box, separately. Then, we use a transformation program to figure out that the apple is on the box by their position information, Finally, the transformation program outputs the proposition $on < apple, box >$ as the state of RMDP. As the state is presented in FOL propositions, the actions of the RMDP are predefined predicates, e.g., $move(x, y)$, each action in MDP is mapped to such a predicate.

The abstraction procedure not only reduces the problem scale, but also incorporates semantic information. All entities and predicates used in the RMDP are predefined in a background knowledge base. As an additional benefit, the information makes the result of our framework more explainable than those end-to-end systems.

3.3 Planning

The reinforcement learning problem is equivalent to a planning problem, if every parameter of the MDP is known. Note that the transition matrix T and reward function R of DRL are generally unknown. That is the reason why we need to run Q-learning instead of value iteration to solve the original task. In RMDP, we present the transition matrix T_r as a set of logic deduction rules and parameters determined by the playing trajectories. Given the input state and action, logic deduction rules generate several alternative next states, and the probability parameter of each next state is calculated from the statistics of the trajectories played by the DRL algorithm. The reward function of the RMDP is a subset of the reward function of the MDP, all non-zero reward state-action pairs of the MDP are remained in the RMDP. To be specific, for any MDP states s_i and s_j and a RMDP state s_r that satisfy $s_r = f_s(s_i) = f_s(s_j)$ and $s_j = T(s_i, a)$, we ensure that $R(s_i, a) = 0$ by the design of f_s. Finally, the

discount factor λ_r and reward function R_r remain the same with MDP.

We adopt the value iteration algorithm to solve the RMDP and obtain the optimal value function of each state. Since the optimal value function is calculated by back propagation, it can be used as an oracle for exploration. Value iteration extends a k-steps-to-go horizon of the value function V_r^k to a (k + 1)-steps-to-go value function V_r^{k+1} using the bellman operator B^*.

$$\begin{aligned} V_r^{k+1}(s_r) =& (B^* V_r^k)(s_r) \\ =& R_r(s_r, a_r, s_r') + \lambda_r \sum_{s_r' \in S_r} T_r(s_r, a_r, s_r') V_r^k(s_r'). \end{aligned} \tag{4}$$

The details of the value iteration algorithm can be described in several steps. It first looks at the state s_r' that is reachable by some actions, for which the value $V_r^k(s')$ is already known. Then, for all $T_r(s_r, a_r, s_r') > 0$, it calculates the summation of their value and multiplication with the discount factor, $\lambda_r \sum_{s_r' \in S_r} T_r(s_r, a_r, s_r') V_r^k(s_r')$. Finally, because only the highest state value is desired, the final value $V_r^{k+1}(s_r)$ maximizes over the set of applicable actions. After the algorithm is executed, we obtain the optimal value function of RMDP. Despite it cannot be accepted as the value function of MDP. it can serve as the potential function in reward shaping to guide the exploration in MDP.

3.4 Feedback

The planning result V_r is utilized to assign credit in DRL through a two-step mechanism—feedback and reward shaping.

At the feedback step, each state s in MDP is assigned a potential function $\phi(s)$, which can be conducted by substituting Eq. (3) into Eq. (4) as:

$$\begin{aligned} \phi(s) &= V_r(f_s(s)) \\ &= V_r(s_r). \end{aligned} \tag{5}$$

At the reward shaping step, we introduce R' to represent the reward function used in DRL after reward shaping. It is defined as:

$$R'(s, a, s') = R(s, a, s') + F(s, a, s'). \tag{6}$$

The potential function ϕ is not the optimal value function for the state s, so it cannot be used to derive the optimal policy directly. However, as (Ng, Harada, and Russell, 1999) proposed, $\phi(s)$ can transform the long-term decision making problem into a short-term decision making problem, which can accelerate the learning process. Thus, the function F is calculated by the potential function ϕ as

$$F(s, a, s') = \gamma \phi(s') - \phi(s). \tag{7}$$

Substituting Eq. (5) and Eq. (7) into Eq. (6), we get the final reward function of M' as

$$R(s, a, s') = R(s, a, s') + \lambda V_r(f_s(s')) - V_r(f_s(s)). \tag{8}$$

As we can see, for a given state-action pair in DRL, if the original reward function $R(s, a, s')$ is zero, Eq. (8) will assign credit for it to guide the learning process.

The rationality of our algorithm lies on the fact that the potential-based reward shaping can ensure that the optimal value function under a modified reward function is also optimal for the original (Ng, Harada, and Russell, 1999), which can theoretically accelerate the converge speed of RL algorithms largely (Arjona-Medina et al., 2018).

4 Application

In this section, we deploy our framework on a complex 3D video game (Doom) to demonstrate that the performance of deep reinforcement learning can be significantly improved by boosting credit assignment with RMDP. We choose Doom as the test scenario since it exhibits a multitude of challenges compared with 2D games (e.g., Go and Atari), especially because the original reward (killing the enemy) of Doom is pretty sparse. It makes the credit assignment and handcrafted reward shaping on Doom pretty hard. Although the policy improvement theorem (Sutton and Barto, 1998) ensures that the algorithm will converge to the optimal solution without reward shaping, however, in real scenarios, it is impractical under limited computation and storage budgets.

We implement our framework based on ViZDoom (Kempka et al., 2016), which is a research platform that builds on top of the Doom game. On ViZDoom, researchers are able to access the screen buffer and game variables as inputs, and perform actions as outputs.

4.1 Architecture

The instantiated framework of our method for Doom is shown in Fig. 2, which incorporates the potential function derived RMDP into the A3C algorithm (Mnih et al., 2016). Note that, our framework is agnostic to the DRL algorithm, thus we adopt A3C instead of DQN due to that it achieves the best performance on Doom. The bottom part demonstrates the general working procedure of deep reinforcement learning. The A3C algorithm takes the screen buffer, game variables from the environment as inputs, and outputs corresponding actions. Due to the complexity, traditional methods can only pick up several state-action pairs to assign credits (Wu and Tian, 2017), while our method is able to assign non-zero credit for all the incoming state-action pairs if the potential function value of the current state and next state is not the same. The upper part represents the implementation of abstraction and logic planning procedure of the RMDP. The abstraction procedure of Doom consists of two components, object detection, and localization, generating 2D and 3D information correspondingly. Object detection generates bounding boxes, each one of them indicates for an entity. Localization generates the location information, which figures out the spatial relationship between the entities. The extracted entities and related information are fed to the predefined predicates to obtain the corresponding state in RMDP. Details of the predefined predicates are discussed in Sec. 4.3. Recall that, since every parameter of the RMDP is known, it is solved by planning algorithm offline, and the result is saved in the potential function. The original reward of the A3C model is combined with the approximate value

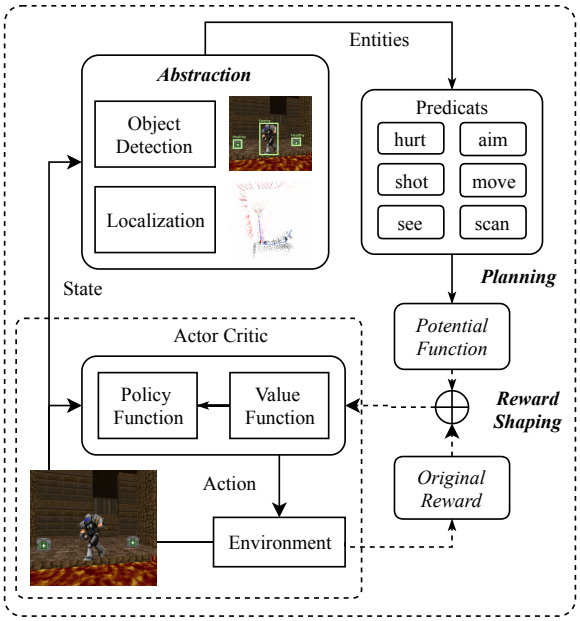

Figure 2: In our framework, deep reinforcement learning and relational reinforcement learning are working in a parallel way. On each action of the agent, the received state is passed both to the abstraction and actor-critic module. The abstraction module transfers the raw state into entities and feeds them to predefined predicates to obtain the relational state, which is substituted to the potential function to obtain the approximate value of the state. Finally, the value of the relation state is combined with the original reward by reward shaping and feedback to the agent.

which is obtained by substituting the relation state in potential function. Finally, the reshaped reward is fed to the agent as training signals.

4.2 Abstraction

Abstraction aims to map the state expressed in the raw pixel into to FOL predicates that the RMDP can access. It is pretty trivial for board games such as "Go" since the state can be represented as a 2D array with length 361. However, the state space would be orders of magnitude larger for 3D games, which makes it difficult to conduct effective planning. To address this issue, we propose to recognize the most relevant components that may facilitate the subsequent logic reasoning procedure in the environments.

We adopt object detection to extract entities and localization to obtain their relationships. The object detection neural network (Ren et al., 2015) generates the bounding box as outputs. Each bounding box indicates for an entity, including enemies, weapons, health kits, armors and other useful items. We apply ORB-SLAM (Mur-Artal, Montiel, and Tardos, 2015) on ViZDoom to build the map and localize the agent itself, which is able to specify the relationship between objects by providing the coordinates. Based on the result of object detection and localization, we are able to transform the raw pixel input into predicates like "hurt" or "move" shown in Tab. 1. Specifically, the doom game provides game variables to represent the current state of the agent, e.g. health, armors, and weapons. Each time, we feed the most recent 4 frames with corresponding game variables to the abstraction module, so it can detect the health decrease by comparing the game variable. Similarly, "move" is detected by tracking the position of the agent, or comparing the changes of the enemy bounding boxes between adjacent frames.

4.3 Definition of the Relational MDP

In a death-match, all participants in Doom are hostile to each other. The goal is to kill the enemies as many times as possible within a given time. The final rankings are determined by *frags*, which is defined as

$$frags = num(Kill) - num(Suicide), \qquad (9)$$

where $num(Kill)$ and $num(Suicide)$ denote the number of the enemies killed by the agent and the number of the agent suicides, respectively.

All activities happened in the death-match can be described as relationship between entries, e.g. moving and pick up resources. In this case, we define three types of entities in Doom, i.e., the agent which is controlled by us, the opponents which are controlled by other participants, and resource items that can be picked up. Their relationship is expressed as state predicates, which describe the status of an agent, e.g., dead or alive. Action predicates represent the abstracted actions that the agent can take, e.g., move, shoot, etc. We summarize all predicates of Doom in Table 1. From an analogy perspective, predicates listed in 1 define the "topological" relationship between objects, which ignores the specific values of parameters like distance or direction to capture the most fundamental attributes of the game.

Table 1: Entities and predicates in Doom.

Entities	agent	the agent controlled by us
	opponent	other agents
	items	pickable resources
State Predicate	hurt(x)	x has a health decrease
	see(x,y)	x is able to see y
	aimed(x,y)	x is targeted by y
	move(x)	x is moving
	stay(x)	x is standing still
	alive(x)	x is alive
	dead(x)	x is dead
Action Predicate	approach(x, y)	x walk to y
	keepmove(x)	x keep moving
	scan(x)	x checks around
	aim(x,y)	x aim at y by turning
	shoot(x)	x fires the weapon

For example, an image with two health kits and one enemy that shown in Fig 2 can be expressed as $\{see < agent, healthkit_0 >, see < agent, healthkit_1 >, aim < agent, enemy_0 >, alive < agent >, alive < enemy_0 >\}$. Note that, in order to limit the number of all valid states, the exact position and direction of the agent are not expressed in

the states. For the same reason, the information of weapon, armor, and ammo that the agent currently holds are also omitted, and the number of enemies is limited to 3 at the most. According to our test, it is easy for the agent to learn the behavior of picking up resource items.

5 Evaluation

We evaluate our framework by conducting various experiments on the Doom game. For each experiment, we evaluate the performance in terms of two metrics: (1) *frags*, which is equal to the number of killings minus the number of suicides; (2) *death*, which counts the number of times being killed by other agents. By integrating the derived potential-based reward function to the A3C algorithm, we obtain substantial improvements in terms of both metrics.

We first analyze the agent performance in an ablation study, then we compare our agent with two state-of-the-art agents, F1 (Wu and Tian, 2017) and IntelAct (Dosovitskiy and Koltun, 2016), and finally report the result of attending a competition of Doom. In all of our experiments, we simply put all trajectories into a fixed sized replay memory and manage it in an first-in-first-out style. The training batches are sampled from the replay memory, and the batch size is set as 128, the discount factor is $\gamma = 0.99$, and the learning rate is $\alpha = 10^{-4}$. Note that the length of each trajectory is determined by the death time of the agent, which makes it not a fixed number.

5.1 Ablation Study

As we mentioned in the previous sections, since the goal of Doom is to kill as many enemies as possible, the original reward function only assigns credits for the last shoot, otherwise, it is always zero. In this section, we provide an ablation study by summarizing the derived credit assignment into two categories by their meaning, namely attack and defense. Each of them is expressed as a setting of the reward function. Recalling the function $F(s, a, s')$ used in Eq. 6, we obtain the attack reward function by only reserving the value of F when the action is shooting, otherwise setting it to zero. Similarly, the defense reward function can be obtained by reserve value of F when the action is moving. Then, we adopt those reward function setting to train and test the agent individually to measure the effects of different credit assignments.

Attack If there exists any enemy currently in view, the agent should aim at it and fire immediately. If there exist more than one enemies, the agent should pick the nearest one.

Defense If the agent is targeted by an enemy, it should move to get rid of it. Doom does not provide defense instruments or terrains, e.g., shield or fortifications. Thus, the only method of defense is to increase the difficulty of enemy attacks, and the continuous movement of the agent will decrease the accuracy of the enemy shot.

Note that all the aforementioned credit assignment categories are apparent for a human user if one understands the goal and regulations of the game. However, it is non-trivial for an RL agent to learn the strategies based on the trial-and-error interactions with a dynamic environment. It would take an agent numerous trials to grasp such strategies via deep reinforcement learning, while the concept is more accessible via relational reinforcement learning. Since the credit assignment is derived from the abstracted RMDP, the risk of introducing undesirable bias to the learning process is reduced to a minimum.

Table 2: Frags and death number of three versions of our agent, which are trained with different credit assignment.

	FlatMap		Map01		Map02	
	frags	death	frags	death	frags	death
A3C	288	**9**	14	**41**	16	**41**
Attack	859	22	291	122	**256**	90
Defense	845	26	186	101	95	87
All	**916**	23	**333**	117	224	84

By applying those credit assignment settings independently on A3C, we obtain four different versions of the agent, "Original A3C", "Attack enhanced", "Defense enhanced" and the complete version of our agent with "All enhancements". As shown in Table 2, the A3C version without any enhancement obtains both the least frags and least deaths, because its number of battles is much less than the other versions. Applying the attack enhancement on A3C improves the attack efficiency. On all maps, the number of frags of attack enhancement is larger than the defense enhancement version, especially on Map01 and Map02, as they provide more powerful weapons. On Map01, the effect of defense enhancement is most obvious. The reason is that the flying process of the rocket fired by the rocket launcher gives more dodge opportunities compared with other weapons. Applying all enhancements together enables the agent to live longer and attack more efficiently in a fight. It demonstrates that the all enhancement version achieves higher frags with fewer or equal deaths, which implies that its winning rate is higher than the other two single enhancement versions.

Table 3: Frags and death number of F1, InteAct and ours

	FlatMap		Map01		Map02	
	frags	death	frags	death	frags	death
F1	451	**9**	282	**110**	35	**56**
IntelAct	864	15	-86	164	134	105
Ours	**916**	23	**333**	117	**224**	84

5.2 Comparing with Known Opponents

We also compare the performance of our agent with two state-of-the-art methods. One is F1 (Wu and Tian, 2017) which is the champion of the known-map-track of ViZDoom competition 2016, and InteAct (Dosovitskiy and Koltun, 2016) who won the champion of the unknown-map-track of ViZDoom competition 2016. We chose FlatMap, Map01 and Map02 as test maps according to the following reason. FlatMap eliminates the random factor caused by terrain, Map01 is the competition map of Track1 with rocket

Table 4: Results on a Doom AI Competition.

Agent	1	2	3	4	5	6	7	8	9	10	Total
1st	15	20	25	24	18	**32**	**42**	**42**	**32**	25	275
ours	**24**	**31**	**31**	**35**	**34**	21	28	28	21	20	273
3rd	15	17	17	18	22	26	21	25	18	14	193
4th	4	3	25	21	11	8	25	20	26	21	164
5th	10	9	3	13	17	16	7	6	27	**31**	139

launcher as the only available weapon, and Map02 is one of the competition maps of Track2 with all weapons available. For each agent, we evaluate 30 rounds of death-match with 2 minutes per round. All agents are tested by combating with 15 build-in Doom bots on three maps.

Results are reported in Table 3. F1 performs well on Map01, while for the other two maps, it demonstrates an obvious performance degeneration. We speculate that F1 was trained overfitting to Map01 to maximize their performance on Track1. In contrast, the ability to predict future values of game variables helps IntelAct to handle all kinds of weapons and ammo, yielding better performance on FlatMap and Map02 than F1. The weakness of IntelAct is that the map terrain affects its performance dramatically. Since Map01 is much narrow than Map02, at most of the time, IntelAct tends to stuck on some corner of the map or just stood somewhere and look around, which makes it is easy to be killed by other agents. Furthermore, due to the blast effect of the rocket, IntelAct has experienced a lot of suicides, which makes the frags negative.

The performance of those reward shaping methods is quite dependent on their training experience. In contrast, the potential function in our framework is derived from the Relational MDP, which are not dependent on any scenario features, e.g., special weapon or terrain. It facilitates our agent to demonstrate more general capabilities in various environments. The results show that our agent outperforms F1 and IntelAct by a large margin on all maps. On the death number, F1 is the least in all agents. Our agent chooses to involve in more combats to get higher frags, this strategy also increases the death number a little.

5.3 Real World Competition

We report the results of participating in an online Doom competition fighting with unknown opponents on unknown maps, to obtain a real-world test of our method. The competition consists of 10 Ten-Minute rounds. The final rank is determined by the total frags number. At the beginning of each round, all agents were born in different places on the same map with unknown terrains. Once the agent was killed in the game, there will be a reborn delay of 10 seconds as a punishment. It thus encourages an agent to shoot as many enemies as possible while staying alive to maintain more valid game time.

Table 4 shows the frags result of the top 5 contestants of the competition. Our agent won the first place in 5 rounds among all the 10 rounds. We achieved the 2nd place in final, slightly behind the first place but gained 23.5% higher score than the third place. This further demonstrates that our agent is endowed with powerful capabilities under the guidance of the derived credit assignment.

Table 5: Detailed Combating Statistics

	1st	ours	3rd	4th	5th	6th	7th
Kills	**275**	**275**	241	216	181	146	84
Death	220	**186**	247	195	200	240	221
Suicide	**0**	2	20	23	17	7	22

Table 5 shows details of the competition. With a solid performance, our agent achieves the equally maximum number of kills and least number of deaths at the same time. From the released competition videos, our agent is able to shoot on the enemies accurately once they show up in our sight and search for the enemies when being attacked. On the defense part, benefiting from the derived potential function, our agent keeps moving most of the time and seldom gets stuck. The high speed moving makes it difficult for the opponent to aim at it, and also increases the chance of exploring the unknown regions to get supplies, e.g., health kits and armors.

Our agent has undesirable suicides, which can be further improved by incorporating image segmentation in the abstraction procedure. Suicide may result from a bomb explosion of its own weapons, such as rockets, or special terrain, e.g., walking on lava. Avoiding suicides requires to deal with special terrain and weapon features. In order to maintain the versatility of our framework, we do not include targeted categories to handle those situations currently.

6 Conclusions

In this paper, we propose a new framework to assign credits for non-terminal state-action pairs in an automated manner, in order to accelerate the learning process of the deep reinforcement learning. In particular, we introduce relational reinforcement learning as a compact representation of deep reinforcement learning. And we solve it to derive the optimal value function, which is then used as the potential function to assign credits for state-action pairs in deep reinforcement learning. Comparing with previous credit assignment methods, our framework not only accelerates the training speed but also improves the final performance. Evaluation on the Doom game has shown that our method has improved the performance significantly, which manifests in outperforming previous state-of-the-art agents and winning a runner-up in a Doom competition.

References

Arjona-Medina, J. A.; Gillhofer, M.; Widrich, M.; Unterthiner, T.; and Hochreiter, S. 2018. Rudder: Re-

turn decomposition for delayed rewards. *arXiv preprint arXiv:1806.07857*.

Dosovitskiy, A., and Koltun, V. 2016. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine learning* 43(1-2):7–52.

Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; and Jaśkowski, W. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*.

Kulkarni, T. D.; Saeedi, A.; Gautam, S.; and Gershman, S. J. 2016. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*.

Lample, G., and Chaplot, D. S. 2017. Playing fps games with deep reinforcement learning. In *AAAI*, 2140–2146.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

Mur-Artal, R.; Montiel, J. M. M.; and Tardos, J. D. 2015. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics* 31(5):1147–1163.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, 278–287.

OpenAI. 2018a. The international 2018: Results. `https://blog.openai.com/the-international-2018-results/`.

OpenAI. 2018b. Openai five. `https://blog.openai.com/openai-five/`.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Sutton, R. S., and Barto, A. G. 2017. Reinforcement learning: An introduction.

Wu, Y., and Tian, Y. 2017. Training agent for first-person shooter game with actor-critic curriculum learning. In *Submitted to International Conference on Learning Representations*.

Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; et al. 2018. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*.