Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Learning to Design Games: Strategic Environments in Reinforcement Learning

Haifeng Zhang[1*], **Jun Wang**[2], **Zhiming Zhou**[3], **Weinan Zhang**[3], **Ying Wen**[2], **Yong Yu**[3], **Wenxin Li**[1]
[1] Peking University
[2] University College London
[3] Shanghai Jiao Tong University
pkuzhf@pku.edu.cn, jun.wang@cs.ucl.ac.uk, wnzhang@sjtu.edu.cn

Abstract

In typical reinforcement learning (RL), the environment is assumed given and the goal of the learning is to identify an optimal policy for the agent taking actions through its interactions with the environment. In this paper, we extend this setting by considering the environment is not given, but controllable and learnable through its interaction with the agent at the same time. This extension is motivated by environment design scenarios in the real-world, including game design, shopping space design and traffic signal design. Theoretically, we find a dual Markov decision process (MDP) w.r.t. the environment to that w.r.t. the agent, and derive a policy gradient solution to optimizing the parametrized environment. Furthermore, discontinuous environments are addressed by a proposed general generative framework. Our experiments on a Maze game design task show the effectiveness of the proposed algorithms in generating diverse and challenging Mazes against various agent settings.

1 Introduction

Reinforcement learning (RL) is typically concerned with a scenario where an agent (or multiple agents) taking actions and receiving rewards from an environment [Kaelbling *et al.*, 1996], and the goal of the *learning* is to find an optimal policy for the agent that maximizes the cumulative reward when interacting with the environment. Successful applications include playing games [Mnih *et al.*, 2013; Silver *et al.*, 2016], scheduling traffic signal [Abdulhai *et al.*, 2003], regulating ad bidding [Cai *et al.*, 2017], to name just a few.

In most RL approaches, such as SARSA and Q-learning [Sutton and Barto, 1998], the model of the environment is, however, not necessarily known a priori before learning the optimal policy for the agent. Alternatively, model-based approaches, such as DYNA [Sutton, 1990] and prioritized sweeping [Moore and Atkeson, 1993], require establishing the environment model while learning the optimal policy. Nonetheless, in either case, the environment is assumed given and mostly either stationary or non-stationary without a purposive control [Kaelbling *et al.*, 1996].

In this paper, we extend the standard RL setting by considering the environment is strategic and controllable. We aim at learning to design an environment via interacting with an also learnable agent or multiple agents. This has many potential applications, ranging from designing a game (environment) with a desired level of difficulties in order to fit the current player's learning stage [Togelius and Schmidhuber, 2008] and designing shopping space to impulse customers purchase and long stay [Penn, 2005] to controlling traffic signals [Ceylan and Bell, 2004]. In general, we propose and formulate the design problem of environments which interact with intelligent agents/humans. We consider designing these environments via machine learning would release human labors and benefit social efficiency. Comparing to the well-studied image design/generation problem [Goodfellow *et al.*, 2014], environment design problem is new in three aspects: (i) there is no ground-truth samples; (ii) the sample to be generated may be discontinuous; (iii) the evaluation of a sample is through learning intelligent agents.

Our formulation extends the scope of RL by focusing on the environment modeling and control. Particularly, in an adversarial case, on one hand, the agent aims to maximize its accumulative reward; on the other hand, the environment tends to minimize the reward for a given optimal policy from the agent. This effectively creates a minimax game between the agent and the environment. Given the agent's playing environment MDP, we, theoretically, find a dual MDP w.r.t. the environment, i.e., how the environment could decide or sample the successor state given the agent's current state and an action taken. Solving the dual MDP yields a policy gradient solution [Williams, 1992] to optimize the parametric environment achieving its objective. When the environment's parameters are not continuous, we propose a generative modeling framework for optimizing the parametric environment, which overcomes the constraints on the environment space. Our experiments on a Maze game generation task show the effectiveness of generating diverse and challenging Mazes against various types of agents in different settings. We show that our algorithms would be able to successfully find the weaknesses of the agents and play against them to generate purposeful environments.

The main contributions of this paper are threefold: (i) we propose the environment design problem, which is novel and

[*] This work is done during Haifeng Zhang's visit at UCL. Jun Wang and Weinan Zhang are the corresponding authors of this paper.

potential for practical applications; (ii) we reduce the problem to the policy optimization problem for continuous cases and propose a generative framework for discontinuous cases; (iii) we apply our methods to Maze game design tasks and show their effectiveness by presenting the generated non-trivial Mazes.

2 Related Work

Reinforcement learning (RL) [Sutton and Barto, 1998] studies how an intelligent agent learns to take actions through the interaction with an environment over time. In a typical RL setting, the environment is unknown yet fixed, and the focus is on optimizing the agent policies. Deep reinforcement learning (DRL) is a marriage of deep neural networks [LeCun *et al.*, 2015] and RL; it makes use of deep neural networks as a function approximator in the decision-making framework of RL to achieve human-level control and general intelligence [Mnih *et al.*, 2015]. In this paper, instead, we consider a family of problems that is an extension of RL by considering that the environment is controllable and strategic. Unlike typical RL, our subject is the strategic environment not the agent, and the aim is to learn to design an optimal (game) environment via the interaction with the intelligent agent.

Our problem of environment design is related to the well-known mechanism design problem[Nisan and Ronen, 2001], which studies how to design mechanisms for participants that achieves some objectives such as social welfare. In most studies, the designs are manual. Our work focuses on automated environment (mechanism) design by machine learning. Thus, we formulate the problem based on MDP and provide solutions based on RL. In parallel, the automated game-level design is a well-studied problem by applying search-based procedural content generation[Togelius *et al.*, 2011]. For generating game-levels that conform to design requirements, genetic algorithm (GA) is proposed as a searcher. Our work instead providing sound solutions based on RL methods, which bring new properties such as gradient direction searching and game feature learning.

In the field of RL, our problem is related to safe/robust reinforcement learning, which maximizes the expectation of the return under some safety constraints such as uncertainty [Garcıa and Fernández, 2015; Morimoto and Doya, 2005], due to the common use of parametric MDPs. However, our problem setting is entirely different from safe RL as their focus is on single agent learning in an unknown environment, whereas our work is concerned with the learning of the environment to achieve its own objective. Our problem is also different from agent reward design [Sorg *et al.*, 2010], which optimizes designer's cumulative reward given by a fixed environment (MDP). However, the environment is learnable in our setting. Another related work, FeUdal networks [Vezhnevets *et al.*, 2017], introduces transition policy gradient to update the proposed manager model, which is a component of agent policy. This is different from our transition gradient which is for updating the environment.

Our formulation is a general one, applicable in the setting where there are multiple agents [Busoniu and De Schutter,]. It is worth mentioning that although multi-agent reinforcement learning (MARL) studies the strategic interplays among different entities, the game (either collaborative or competitive) is strictly among multiple agents [Littman, 1994; Hu and Wellman, 2003]. By contrast, the strategic interplays in our formulation are between an agent (or multiple agents) and the environment. The recent work, interactive POMDPs [Gmytrasiewicz and Doshi, 2005], aims to spread beliefs over physical states of the environment and over models of other agents, but the environment in question is still non-strategic. Our problem, thus, cannot be formulated directly using MARL as the decision making of the environment is in an episode-level, while policies of agents typically operate and update in each time-step within an episode.

In addition, our minimax game formulation can also be found in the recently emerged generative adversarial nets (GANs), where a generator and a discriminator play a minimax adversarial game [Goodfellow *et al.*, 2014]. Compared to GANs, our work addresses a different problem, where the true samples of desired environments are missing in our scenario; the training of our environment generator is guided by the behaviours of the agent (corresponding the GAN discriminator) who aims to maximize its cumulative reward in a given environment.

3 RL with Controllable Environment

3.1 Problem Formulation

Let us first consider the standard reinforcement learning framework. In this framework there are a learning agent and a Markov decision process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ denotes state space, $\mathcal{A}$ action space, $\mathcal{P}$ state transition probability function, $\mathcal{R}$ reward function and γ discounted factor. The agent interacts with the MDP by taking action a in state s and observing reward r in each time-step, resulting in a trajectory of states, actions and rewards: $H_{1\ldots\infty} = \langle S_1, A_1, R_1, S_2, A_2, R_2 \ldots \rangle, S_t \in \mathcal{S}, A_t \in \mathcal{A}, R_t \in \mathbb{R}$, where $\mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] = \mathcal{P}(s, a, s')$ and $\mathbb{E}[R_t | S_t = s, A_t = a] = \mathcal{R}(s, a)$ hold.[1] The agent selects actions according to a policy π_ϕ, where $\pi_\phi(a|s)$ defines the probability that the agent selects action a in state s. The agent learns π_ϕ to maximize the return (cumulative reward) $G = \sum_{t=1}^{\infty} \gamma^{t-1} R_t$.

In the standard setting, the MDP is given fixed while the agent is flexible with its policy to achieve its objective. We extend this setting by also giving flexibility and purpose to $\mathcal{M}$. Specifically, we parametrize $\mathcal{P}$ as $\mathcal{P}_\theta$ and set the objective of the MDP as $O(H)$, which can be arbitrary based on the agent's trajectory. We intend to design (generate) an MDP that achieves the objective along with the agent achieving its own objective:

$$\theta^* = \arg\max_\theta \mathbb{E}\big[O(H) | \mathcal{M}_\theta = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\theta, \mathcal{R}, \gamma \rangle; \pi_{\phi^*} = \arg\max_{\pi_\phi} \mathbb{E}[G | \pi_\phi; \mathcal{M}_\theta]\big]. \quad (1)$$

Adversarial Environment

In this paper, we consider a particular objective of the environment that it acts as an adversarial environment minimizing the expected return of the single agent, i.e., $O(H) =$

[1] In this paper, we use S_t, A_t, R_t when they are in trajectories while using s, a, r otherwise.

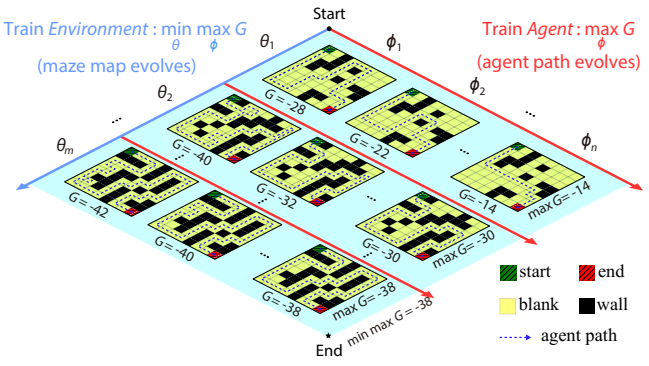

Figure 1: An example of adversarial Maze design. The detailed definition of the Maze environment is provided in Sec.4. In short, an agent tries to find the shortest path from the start to the end in a given Maze map, while the Maze environment tries to design a map to make the path taken by the agent longer. In the direction of ϕ, the parameter of an agent policy evolves, whereas in the direction of θ, the parameter of the Maze environment evolves. The cumulative reward G is defined as the opposite number of the length of the path.

$\sum_{t=1}^{\infty} -\gamma^{t-1} R_t = -G$. This adversarial objective is useful in the game design domain because for many games the game designer need to design various game levels or set various game parameters to challenge game players playing with various game strategies. Thus, the relationship between the environment(game) and the agent(player) are adversarial. We intend to transfer this design work from human to machine by applying appropriate machine learning methods. Formally, the objective function is formulated as:

$$\theta^* = \arg\min_{\theta} \max_{\phi} \mathbb{E}[G|\pi_\phi; \mathcal{M}_\theta = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\theta, \mathcal{R}, \gamma \rangle]. \quad (2)$$

In general, we adopt an iterative framework for learning θ and ϕ. In each iteration, the environment updates its parameter to maximize its objective w.r.t. the current agent policy then the agent updates its policy parameter by taking sufficient steps to be optimal w.r.t. the updated environment, as illustrated by Fig. 1 for learning the environment of a Maze. Since the agent's policy can be updated using well-studied RL methods, we focus on the update methods for the environment. In each iteration, given the agent's policy parameter ϕ^*, the objective of the environment is

$$\theta^* = \arg\min_{\theta} \mathbb{E}[G|\mathcal{M}_\theta = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\theta, \mathcal{R}, \gamma \rangle; \pi_{\phi^*}]. \quad (3)$$

In the following sections, we propose two methods to solve this problem for continuous and discontinuous environments.

3.2 Gradient Method for Continuous Environment

In this section, we propose a gradient method for continuous environment, i.e. the value of the transition probability for any $\langle s, a, s' \rangle$ can be arbitrary in $[0, 1]$. Thus, the parameter θ of the environment actually consists of the values of the transition function $\mathcal{P}(s, a, s')$ for each $\langle s, a, s' \rangle$. Our task is to optimize the values of the transition function to minimize the agent's cumulative reward.

To update the environment, we try to find the gradient of the environment objective w.r.t. θ. We derive the gradient by taking a new look at the environment and the agent in the opposite way, that the original environment $\mathcal{M}^A$ as an agent and the original agent as a part of the new environment $\mathcal{M}^E$. Viewing in this way, the original environment $\mathcal{M}^A$ takes action A_t^E to determine the next state S_{t+1}^A given the current state S_t^A and the agent's action A_t^A. Thus we define the state s^E in $\mathcal{M}^E$ as the combination $\langle s^A, a^A \rangle$. On the other hand, given the original environment's action $A_t^E = S_{t+1}^A$, the agent policy $\pi_{\phi^*}^A(s^A)$ acts as a transition in $\mathcal{M}^E$ to determine A_{t+1}^A as part of the next state $S_{t+1}^E = \langle S_{t+1}^A, A_{t+1}^A \rangle$ in $\mathcal{M}^E$. Furthermore, optimizing agent policy in $\mathcal{M}^E$ is equal to optimizing environment transition in $\mathcal{M}^A$.

Theoretically, we reduce our transition optimization problem in Eq. (3) to the well-studied policy optimization problem through a proposed concept of a duel MDP-policy pair.

Definition 1 (Duel MDP-policy pair)**.** *For any MDP-policy pair $\langle \mathcal{M}^A, \pi^A \rangle$, where $\mathcal{M}^A = \langle \mathcal{S}^A, \mathcal{A}^A, \mathcal{P}^A, \mathcal{R}^A, \gamma^A \rangle$ with start state distribution p_1^A and terminal state set $\mathcal{S}_T^A$, there exists a dual MDP-policy pair $\langle \mathcal{M}^E, \pi^E \rangle$, where $\mathcal{M}^E = \langle \mathcal{S}^E, \mathcal{A}^E, \mathcal{P}^E, \mathcal{R}^E, \gamma^E \rangle$ with start state distribution p_1^E and terminal action set $\mathcal{A}_T^E$ satisfying:*

- $\mathcal{S}^E = \mathcal{S}^A \times \mathcal{A}^A = \{\langle s^A, a^A \rangle | s^A \in \mathcal{S}^A, a^A \in \mathcal{A}^A\}$, *a state in $\mathcal{M}^E$ corresponds to a combination of successive state and action in $\mathcal{M}^A$;*
- $\mathcal{A}^E = \mathcal{S}^A = \{s^A | s^A \in \mathcal{S}^A\}$, *an action in $\mathcal{M}^E$ corresponds to a state in $\mathcal{M}^A$;*
- $\mathcal{P}^E(s_i^E, a^E, s_{i'}^E) = \mathcal{P}^E(\langle s_j^A, a_k^A \rangle, s^A, \langle s_{j'}^A, a_{k'}^A \rangle) = \begin{cases} \pi^A(a_{k'}^A | s^A) & s^A = s_{j'}^A \\ 0 & s^A \neq s_{j'}^A \end{cases}$, *the transition in $\mathcal{M}^E$ depends on the policy in $\mathcal{M}^A$;*
- $\mathcal{R}^E(s_i^E, a^E) = \mathcal{R}^E(\langle s_j^A, a_k^A \rangle, s^A) = \mathcal{R}^A(s_j^A, a_k^A)$, *the rewards in $\mathcal{M}^E$ are the same as in $\mathcal{M}^A$;*
- $\gamma^E = \gamma^A$, *the discounted factors are the same;*
- $p_1^E(s^E) = p_1^E(\langle s^A, a^A \rangle) = p_1^A(s^A)\pi^A(a^A | s^A)$, *start state distribution in $\mathcal{M}^E$ depends on start state distribution and the first action distribution in $\mathcal{M}^A$;*
- $\mathcal{A}_T^E = \{s^A | s^A \in \mathcal{S}_T^A\}$, *terminal action in $\mathcal{M}^E$ corresponds to terminal state in $\mathcal{M}^A$;*
- $\pi^E(a^E | s^E) = \pi^E(s_{i'}^A | \langle s_i^A, a^A \rangle) = \mathcal{P}^A(s_i^A, a^A, s_{i'}^A)$, *policy in $\mathcal{M}^E$ corresponds to transition in $\mathcal{M}^A$.*

We can see that the dual MDP-policy pair in fact describes an equal mechanism as the original MDP-policy pair from another perspective. Based on the dual MDP-policy pair, we give three theorems to derive the gradient of the transition function. The proofs are omitted for space reason.

Theorem 1. *For an MDP-policy pair $\langle \mathcal{M}^A, \pi^A \rangle$ and its duality $\langle \mathcal{M}^E, \pi^E \rangle$, the distribution of trajectory generated by $\langle \mathcal{M}^A, \pi^A \rangle$ is the same as the distribution of a bijective trajectory generated by $\langle \mathcal{M}^E, \pi^E \rangle$, i.e. $\mathbb{P}[H^A | M^A, \pi^A] = \mathbb{P}[H^E | M^E, \pi^E]$, where $H^E = b(H^A), H^A = b^{-1}(H^E)$.*

Theorem 2. *For an MDP-policy pair $\langle \mathcal{M}^A, \pi^A \rangle$ and its duality $\langle \mathcal{M}^E, \pi^E \rangle$, the expected return of two bijective state-action trajectories, $H^A = b^{-1}(H^E)$ from $\langle \mathcal{M}^A, \pi^A \rangle$ and $H^E = b(H^A)$ from $\langle \mathcal{M}^E, \pi^E \rangle$, are equal.*

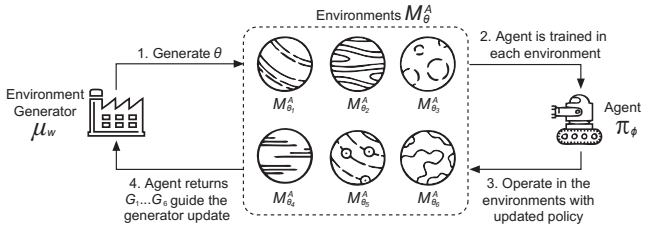

Figure 2: Framework dealing with discontinuous environment. Generator generates environment parameter θ. For each $\mathcal{M}_\theta^A$, the agent policy is trained. Then the policy is tested in the generated environments and the returns are observed, which finally guide the generator to update.

Theorem 3. *For an MDP-policy pair $\langle \mathcal{M}^A, \pi^A \rangle$ and its duality $\langle \mathcal{M}^E, \pi^E \rangle$, the expected return of $\langle \mathcal{M}^A, \pi^A \rangle$ is equal to the expected return of $\langle \mathcal{M}^E, \pi^E \rangle$, i.e., $\mathbb{E}[G^A|\pi^A, \mathcal{M}^A] = \mathbb{E}[G^E|\pi^E, \mathcal{M}^E]$.*

Theorem 2 can be understood by the equivalence between H^A and H^E and the same generating probability of them as given in Theorem 1. Theorem 3 naturally extends Theorem 2 from the single trajectory to the distribution of trajectory according to the equal probability mass function given by Theorem 1.

Now we consider $\langle \mathcal{M}_\theta^A, \pi^A \rangle$ and its duality $\langle \mathcal{M}^E, \pi_\theta^E \rangle$, where $\mathcal{P}_\theta^A$ and π_θ^E are of the same form about θ. Given θ, $\mathcal{P}_\theta^A$ and π_θ^E are exactly the same, resulting in $\mathbb{E}[G^A|\pi^A, \mathcal{M}_\theta^A] = \mathbb{E}[G^E|\pi_\theta^E, \mathcal{M}^E]$ according to Theorem 3. Thus optimizing $\mathcal{P}_\theta^A$ as Eq. (3) is equivalent to optimizing π_θ^E:

$$\theta^* = \arg\min_\theta \mathbb{E}[G|\mathcal{M}_\theta^A; \pi_{\phi^*}^A] = \arg\min_\theta \mathbb{E}[G|\pi_\theta^E; \mathcal{M}_{\phi^*}^E]. \tag{4}$$

We then apply the policy gradient theorem [Sutton *et al.*, 1999] on π_θ^E and derive the gradient for $\mathcal{P}_\theta^A$:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}[\nabla_\theta \log \pi_\theta^E(a^E|s^E) Q^E(s^E, a^E)|\pi_\theta^E; \mathcal{M}_{\phi^*}^E] \\ &= \mathbb{E}[\nabla_\theta \log \mathcal{P}_\theta^A(s_i^A, a^A, s_{i'}^A) V^A(s_{i'}^A)|\mathcal{M}_\theta^A; \pi_{\phi^*}^A],\end{aligned} \tag{5}$$

where $J(\theta)$ is cost function, $Q^E(s^E, a^E)$ and $V^A(s_{i'}^A)$ are action-value function and value function of $\langle \mathcal{M}^E, \pi_\theta^E \rangle$ and $\langle \mathcal{M}_\theta^A, \pi^A \rangle$ respectively; and can be proved equal due to the equivalence of the two MDPs.

We name the gradient in Eq. (5) as *transition gradient*. Transition gradient can be used to update the transition function in an iterative way. In theory, it performs as well as policy gradient since it is equivalent to the policy gradient in the circumstance of the dual MDP-policy pair.

3.3 Generative Framework for Discontinuous Environment

The transition gradient method proposed in the last section only works for continuous environment. For discontinuous environment, i.e. the range of the transition function $\mathcal{P}(s, a, s')$ is not continuous in $[0, 1]$, we cannot directly take the gradient of the transition function w.r.t. θ.

To deal with the discontinuous situation, we propose a generative framework to find the optimal θ alternative to the gradient method. In general, we build a parametrized generator to generate a distribution of the environment, then update the parameter of the generator by evaluating the environments it generates (illustrated in Fig. 2). Specifically, we generate environment parameter θ using a w-parametrized generator μ_w, then optimize w to obtain the (local) optimal w^* and a corresponding optimal distribution of θ. Formally, our optimization objective is formulated as

$$\begin{aligned}w^* = \arg\min_w \mathbb{E}_{\theta\sim\mu_w}\big[\mathbb{E}[G| \\ \mathcal{M}_\theta^A = \langle \mathcal{S}^A, \mathcal{A}^A, \mathcal{P}_\theta^A, \mathcal{R}^A, \gamma^A \rangle; \pi_{\phi^*}]\big].\end{aligned} \tag{6}$$

We model the generation process using an auxiliary MDP $\mathcal{M}^\mu$, i.e., the generator μ_w generates θ and updates w in a reinforcement learning way. The reason we adopt reinforcement learning other than supervised learning is that in this generative task, (i) there is no training data to describe the distribution of the desired environments so we cannot compute likelihood of generated environments and (ii) we can only evaluate a generated environment through sampling, i.e., performing agents in the generated environment and getting a score from the trajectory, which can be naturally modeled by reinforcement learning by viewing the score as a reward of the actions of the generator.

In detail, the generator μ_w consists of three elements $\langle \mathcal{M}^\mu, \pi_w^\mu, f^\mu \rangle$. For generating θ, an auxiliary agent with policy π_w^μ acts in $\mathcal{M}^\mu$ to generate a trajectory H^μ, after that θ is determined by the transforming function $\theta = f^\mu(H^\mu)$, i.e., the distribution of θ is based on the distribution of trajectories, which are further induced by playing π_w^μ in $\mathcal{M}^\mu$. For adversarial environments, the reward of the generator is designed to be opposite to the return of the agent got in $\mathcal{M}_\theta$, which reflects the minimization objective in Eq. (6). Thus, w can be updated by applying policy gradient methods on π_w^μ.

There are various ways to designing $\mathcal{M}^\mu$ for a particular problem. Here we provide a general design that can be applied to any environment. Briefly, we generate the environment parameter in an additive way and ensures the validity along the generation process. In detail, we reshape the elements of θ as a vector $\theta = \langle x_1, x_2, \ldots, x_{N_\theta} \rangle, x_k \in X_k$ and design $\mathcal{M}^\mu = \langle \mathcal{S}^\mu, \mathcal{A}^\mu, \mathcal{P}^\mu, \mathcal{R}^\mu, \gamma^\mu = 1 \rangle$ to generate θ:

- $\mathcal{S}^\mu = \{v_k = \langle x_1, x_2, \ldots, x_k \rangle | k = 0 \ldots N_\theta, \exists v_{N_\theta} = \langle x_1, x_2, \ldots, x_k, x'_{k+1} \ldots x'_{N_\theta} \rangle = \theta, \text{s.t. } \mathcal{P}_\theta^A \in \mathfrak{P}^A\}$;
- $\mathcal{A}^\mu = \bigcup_{k=1\ldots N_\theta} X_k$;
- $\mathcal{P}^\mu$ is defined that for the current state $v_k = \langle x_1, x_2, \ldots, x_k \rangle$ and an action x_{k+1}, if $x_{k+1} \in X_{k+1}$ and $v_{k+1} = \langle x_1, x_2, \ldots, x_{k+1} \rangle \in \mathcal{S}^\mu$ the next state is v_{k+1}, otherwise v_k;
- $\mathcal{R}^\mu$ is defined that for terminal state $v_{N_\theta} = \langle x_1, x_2, \ldots, x_{N_\theta} \rangle = \theta$ the reward is the opposite number of the averaged return got by $\pi_{\phi^*}^A$ acting in $\mathcal{M}_\theta^A$, otherwise the reward is 0.

In addition, the start state is $v_0 = \langle\rangle$ and the terminal states are $v_{N_\theta} = \langle x_1, x_2, \ldots, x_{N_\theta} \rangle$. Corresponding to this $\mathcal{M}^\mu$, $\pi_w^\mu(x_{k+1}|v_k; w)$ is designed to take an action $x_{k+1} \in X_{k+1}$ depending on the previous generated sequence v_k, and the transforming function f^μ is designed as $f^\mu(H^\mu) = v_{N_\theta} = \theta$. Note that due to the definition of $\mathcal{S}^\mu$, any partial parameter v_t without potential to be completed as a valid parameter θ

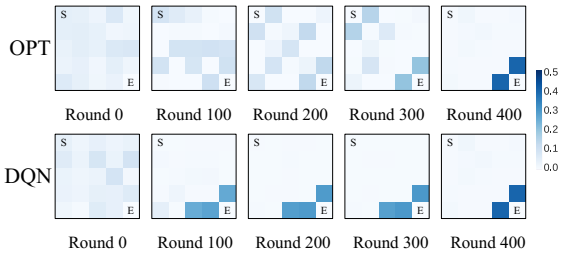

Figure 3: Heatmaps of the blockage probability (soft wall, indicated by the intensity of the color in the cell) distribution throughout 5×5 soft wall Maze learning against the OPT and DQN agents.

is avoided to be generated. This ensures any constraint on environment parameter can be followed. On the other hand, any valid θ is probable to be generated once π_w^μ is exploratory and of enough expression capacity.[2]

4 Experiments with Maze Design

4.1 Experiment Setting

In our experiment, we consider a use case of designing Maze game to test our solutions over the transition gradient method and the generative framework respectively. As shown in both Figs. 4 and 5, the Maze is a grid world containing a map of $n \times n$ cells. In every time-step, the agent is in a cell and has four directional actions $\{N, S, W, E\}$ to select from, and transitions are made deterministically to an adjacent cell, unless there is a *wall* (e.g., the black cells as illustrated in Figs. 4 and 5), in which case no movement occurs. The minimax game is defined as: the agent should go from the north-west cell to the south-east cell using steps as few as possible, while the goal of the Maze environment is to arrange the walls in order to maximize the number of steps taken by the agent.

Note that the above *hard wall* Maze results in an environment that is discontinuous. In order to also test the case of continuous environments, we consider a *soft wall* Maze as shown in Fig. 3. Specifically, instead of a hard wall that completely blocks the agent, each cell except the end cell has a blockage probability (soft wall) which determines how likely the agent will be blocked by this cell when it takes transition action from an adjacent cell. It is also ensured that the sum of blockage probabilities of all cells is 1 and the maximum blockage probability for each cell is 0.5. Thus, the task for the adversarial environment in this case is to allocate the soft wall to each cell to block the agent the most.

Our experiment is conducted on PCs with common CPUs. We implement our experiment environment using Keras-RL [Plappert, 2016] backed by Keras and Tensorflow. [3]

4.2 Results for the Transition Gradient Method

We test the transition gradient method considering the 5×5 soft wall Maze case. We model the transition probability function by a deep convolutional neural network, which is updated by the transition gradient following Eq. (5). We consider the two types of agents: *Optimal (OPT) agent* and *Deep*

[2]The generative framework could also be applied for continuous environment generation although it results in low efficiency comparing to directly updating the environment by gradient.

[3]Our experiment is repeatable and the code is at goo.gl/o9MrDN.

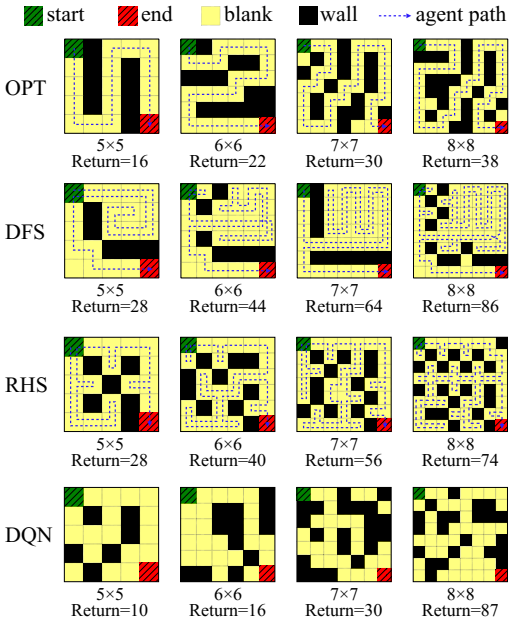

Figure 4: Best Mazes against OPT, DFS, RHS and DQN agents with size ranging from 5×5 to 8×8.

Q-network learning (DQN) agent. The OPT agent has no parameters to learn, but always finds the optimal policy against any generated environment. The DQN agent [Mnih *et al.*, 2013] is a learnable one, in which the agent's action-value function is modeled by a deep neural network, which takes the whole map and its current position as input, processed by 3 convolutional layers and 1 dense layer, then outputs the Q-values over the four directions. For each updated environment, we train the DQN agent to be optimal, as Fig. 1 shows.

Fig. 3 shows the convergence that our transition gradient method has achieved. The change of the learned environment parameters, in the form of blockage probabilities, over time are indicated by the color intensity. Intuitively, the most effective adversarial environment to block the agent is to place two 0.5 soft walls in the two cells next to the end or the beginning cell, as this would have the highest blockage probabilities. We can see that in both cases, using the OPT agent and the DQN agent, our learning method can obtain one of the two most optimal Maze environments.

4.3 Results for Generative Framework

We now test our reinforcement learning generator by the hard wall Maze environment. We follow the proposed general generative framework to design $\mu_w = \langle \mathcal{M}^\mu, \pi_w^\mu, f^\mu \rangle$, which gradually generates walls one by one from an empty map. Particularly, π_w^μ is modeled by a deep neural network that takes an on-going generated map as input and outputs a position for a new wall or a special action for termination. Actions lead to generating walls that completely block the agent are invalid and prevented. We test our generator against four types of agents each on four sizes of maps (from 5×5 to 8×8). Although the objective for every agent is to minimize the number of steps, not every agent has the ability to find the optimal policy because of model restrictions of π_ϕ or limitations in the training phase. Therefore, besides testing our generator against the optimal agent (the OPT agent) and

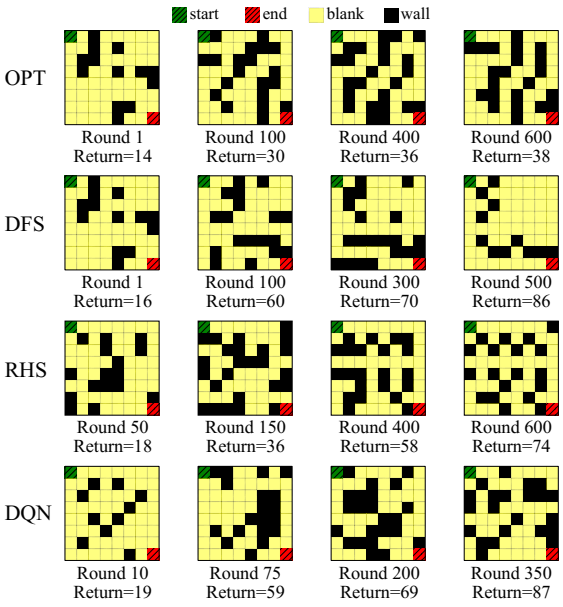

Figure 5: Learning to design Mazes against OPT, DFS, RHS and DQN agents in 8×8 map.

the DQN agent, we also adopt other two imperfect agents for our generator to design specific Mazes in order to understand more about our solution's behaviors. They are:

Depth-first search (DFS) agent. The DFS agent searches the end in a depth-first way. In each time-step, without loss of generality, the DFS agent is set to select an action according to the priority of East, South, North, West. The DFS agent takes the highest priority action that leads to a blank and unvisited cell. If there are none, The DFS agent goes back to the cell from which it comes.

Right-hand search (RHS) agent. The RHS agent is aware of the heading direction and follows a strategy that always ensures its right-hand cell is a wall or the border. In each time-step, (i) the RHS agent checks its right-hand cell, if it is blank, the RHS agent will turn right and step into the cell; (ii) if not, then if the front cell is blank, the RHS agent will step forward; (iii) if the front cell is not blank, the RHS agent will continue turning left until it faces a blank cell, then steps into that cell.

Note that DFS and RHS are designed particularly for discontinuous Mazes. We also limit the network capacity and training time of the DQN agent to make it converge differently from the OPT agent. The learned optimal Mazes are given in Fig. 4 for different agents with different Maze sizes. The strongest Mazes designed by our generator are found when playing against the OPT agent, shown in Fig. 4 (OPT). We see that in all cases, from 5×5 to 8×8, our generator tends to design long narrow paths without any fork, which makes the optimal paths the longest. By contrast, the generator designs many forks to trap the DQN agent, shown in Fig. 4 (DQN), as the DQN agent runs a stochastic policy (ϵ-greedy).

In fact our generator could make use of the weakness from the agents to design the maps against them. Fig. 4 (DFS) shows the results that our generator designs extremely broad areas with only one entrance for the DFS agent to search exhaustively (visit every cell in the closed area twice). Fig. 4 (RHS) shows the Mazes generated to trouble the RHS agent

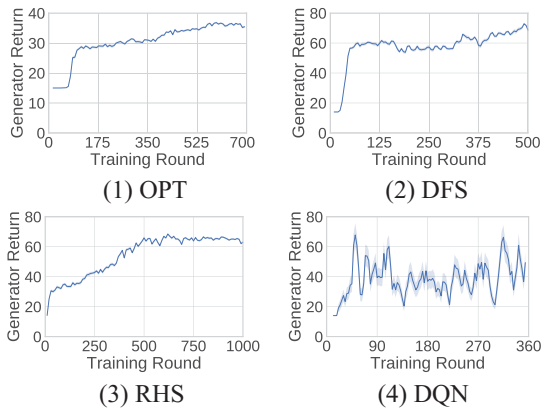

Figure 6: Training curves for OPT, DFS, RHS and DQN agents in 8×8 map. The lines and the shadows show mean and variance of generator return respectively.

the most by creating a highly symmetric Maze.

Next, Fig. 5 shows the snapshots of the results in different learning rounds. They all evolve differently, depending on the types of the agents. For the OPT agent, we find that our generator gradually links isolated walls to form a narrow but long path. For the DFS, our generator gradually encloses an area then broadens and sweeps it in order to best play against the policy that has the priority order of their travel directions. Fig. 5 (RHS) shows that our generator learns to adjust the wall into zigzag shapes to trouble the RHS agent. For the DQN agent, with limited network capacity or limited training time, it is usually the case that it cannot perfectly tell which road to go during the learning. As such, the generator tends to generate many forks to confuse the DQN agent.

Furthermore, Fig. 6 shows the process of training our generator against the four agents in 8×8 map. We find that for OPT, DFS and RHS agents, the generator learns rapidly at first and gradually converges. But for the DQN agent, the learning curve is tortuous. This is because the ability of the DQN agent is gradually improved so it does not accurately and efficiently guide the learning of the generator. Also when the ability of the DQN agent improves greatly and suddenly, the learning curve for the generator may change its direction temporarily. Theoretically, training the DQN agent adequately in each iteration is a promising way towards to monotony and convergence.

5 Conclusions

In this paper, we presented an extension of standard reinforcement learning by considering that the environment is strategic and can be learned. We derived a gradient method by introducing a dual MDP-policy pair for continuous environment. To deal with discontinuous environment, we proposed a novel generative framework using reinforcement learning. We evaluated the effectiveness of our solution by considering designing a Maze game. The experiments showed that our methods can make use of the weaknesses of agents to learn the environment effectively.

In the future, we plan to apply the proposed methods to practical environment design tasks, such as video game design [Hom and Marks, 2007], shopping space design [Penn, 2005] and bots routine planning.

Acknowledgements

This work is financially supported by National Natural Science Foundation of China (61632017) and National Key Research and Development Plan (2017YFB1001904).

References

[Abdulhai *et al.*, 2003] Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 2003.

[Busoniu and De Schutter,] Lucian Busoniu and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning.

[Cai *et al.*, 2017] Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, and Defeng Guo. Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 661–670. ACM, 2017.

[Ceylan and Bell, 2004] Halim Ceylan and Michael GH Bell. Traffic signal timing optimisation based on genetic algorithm approach, including drivers' routing. *Transportation Research Part B: Methodological*, 2004.

[Garcıa and Fernández, 2015] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *JMLR*, 2015.

[Gmytrasiewicz and Doshi, 2005] Piotr J Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *JAIR*, 2005.

[Goodfellow *et al.*, 2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[Hom and Marks, 2007] Vincent Hom and Joe Marks. Automatic design of balanced board games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2007.

[Hu and Wellman, 2003] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine learning research*, 2003.

[Kaelbling *et al.*, 1996] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 1996.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

[Littman, 1994] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, 1994.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[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*, 2015.

[Moore and Atkeson, 1993] Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 1993.

[Morimoto and Doya, 2005] Jun Morimoto and Kenji Doya. Robust reinforcement learning. *Neural computation*, 2005.

[Nisan and Ronen, 2001] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, 2001.

[Penn, 2005] Alan Penn. The complexity of the elementary interface: shopping space. In *Proceedings to the 5th International Space Syntax Symposium*. Akkelies van Nes, 2005.

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

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[Sorg *et al.*, 2010] Jonathan Sorg, Richard L Lewis, and Satinder P Singh. Reward design via online gradient ascent. In *NIPS*, 2010.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[Sutton *et al.*, 1999] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[Sutton, 1990] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*, 1990.

[Togelius and Schmidhuber, 2008] Julian Togelius and Jurgen Schmidhuber. An experiment in automatic game design. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008.

[Togelius *et al.*, 2011] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[Vezhnevets *et al.*, 2017] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.