Partial Consistency for Stabilizing Undiscounted Reinforcement Learning

Haichuan Gao, Zhile Yang, Tian Tan, Tianren Zhang, Jinsheng Ren, Pengfei Sun, Shangqi Guo, and Feng Chen, *Member, IEEE*

***Abstract*— Undiscounted return is an important setup in reinforcement learning (RL) and characterizes many real-world problems. However, optimizing an undiscounted return often causes training instability. The causes of this instability problem have not been analyzed in-depth by existing studies. In this article, this problem is analyzed from the perspective of value estimation. The analysis result indicates that the instability originates from transient traps that are caused by inconsistently selected actions. However, selecting one consistent action in the same state limits exploration. For balancing exploration effectiveness and training stability, a novel sampling method called last-visit sampling (LVS) is proposed to ensure that *a part of actions is selected consistently* in the same state. The LVS method decomposes the state-action value into two parts, i.e., the last-visit (LV) value and the revisit value. The decomposition ensures that the LV value is determined by consistently selected actions. We prove that the LVS method can eliminate transient traps while preserving optimality. Also, we empirically show that the method can stabilize the training processes of five typical tasks, including vision-based navigation and manipulation tasks.**

***Index Terms*— Last visit (LV), partial consistency, reinforcement learning (RL), transient trap, undiscounted return.**

I. Introduction

REINFORCEMENT learning (RL) has achieved outstanding performance in many application fields, such as chess [1] and video games [2]–[4]. Usually, a temporally discounted return is used to train a policy, but an undiscounted return is used to evaluate the final performance of the policy [1], [2], [5]–[7]. Undiscounted returns can directly express criteria such as accumulated scores [2] and success probability [5], which are objectives commonly used in a wide range of tasks, especially tasks in robot control [6], [8]–[11]. For example, warehouse robots are often tasked to reach certain destinations, for which a 5- or 10-min route is equally good. In this case, we may evaluate the robot's task performance by success probability, which could be expressed by a simple undiscounted return about goal-reaching rewards [12].

Although undiscounted returns are considered for task performance, direct optimization of the undiscounted return is often avoided in RL because the optimization leads to training instability, i.e., the performance oscillates during the training process and the training fails to converge [13]–[17]. As a result, discounted returns are commonly used as a surrogate objective during training to stabilize the learning process. However, the discounted return induces a bias to the undiscounted objective, which has been put forward by prior works [17], [18]. Although this bias does not alter the optimal policy learned by the agent when the discount factor is set large enough, the determination of the proper discount factor is difficult, especially for Markov decision processes (MDPs) with long horizons [17], [19]. Thus, many researchers set a discount factor close to 1 to obtain policies with good performances [18].

To illustrate the suboptimality caused by discount factor, a Cliff-Walking task [20] shown in Fig. 1 is taken as an example. This task aims to maximize the probability (MAXPROB [12], [21]) of reaching the goal state marked by a star. In particular, the probability needs to be estimated with the terminal reward that is not temporally discounted. In this task, the agent can follow two paths to the goal state: one path is short but risky (length $= 5$, there are four steps with falling probability $p_{\text{fall}} = 0.1$), and the other is long but safe (length $= 9$). If a commonly used discount factor is adopted, e.g., $\gamma = 0.9$, the learned policy will follow the shorter path rather than the safer one. Although γ satisfying $\gamma^{(9-5)} > (1-p_{\text{fall}})^4$ can produce the optimal policy, the setting of γ requires p_{fall} and the lengths of paths that are task-specific and generally unavailable in complex environments.

The above analysis reveals that undiscounted RL is important but has not received much attention from researchers. Thus, this study aims to directly optimize the undiscounted return and alleviate the training instability problem. The significance of this study is twofold. On the one hand, this study can scale up undiscounted RL to be applied in many fields

Manuscript received 31 August 2021; revised 18 December 2021 and 11 February 2022; accepted 1 April 2022. Date of publication 25 April 2022; date of current version 1 December 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62176133 and Grant 61836004, in part by the Tsinghua–Guoqiang Research Program under Grant 2019GQG0006, and in part by the National Key Research and Development Program of China under Grant 2021ZD0200300. The work of Shangqi Guo was supported by the Shuimu Tsinghua Scholar Program. *(Haichuan Gao and Zhile Yang contributed equally to this work.) (Corresponding authors: Shangqi Guo; Feng Chen.)*

Haichuan Gao, Zhile Yang, Tianren Zhang, Jinsheng Ren, and Pengfei Sun are with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: ghc18@mails.tsinghua.edu.cn; yzl18@mails.tsinghua.edu.cn; zhang-tr19@mails.tsinghua.edu.cn; rjs17@mails.tsinghua.edu.cn; spf16@mails.tsinghua.edu.cn).

Tian Tan is with the Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: tiantan@stanford.edu).

Shangqi Guo is with the Department of Automation and the Department of Precision Instrument, Tsinghua University, Beijing 100086, China (e-mail: shangqi_guo@foxmail.com).

Feng Chen is with the Center for Brain-Inspired Computing Research, Department of Automation, Tsinghua University, Beijing 100190, China, and also with the LSBDPA Beijing Key Laboratory, Beijing 100084, China (e-mail: chenfeng@mail.tsinghua.edu.cn).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2022.3165941.

Digital Object Identifier 10.1109/TNNLS.2022.3165941

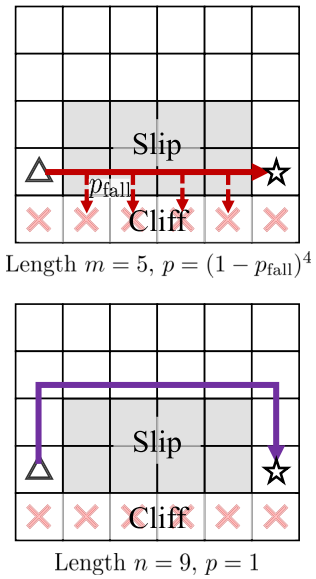

Fig. 1. Cliff-Walking example.

for preserving the optimal policy [13], [16], [17], [22]–[24]. On the other hand, similar to the cases with undiscounted returns [13], [16], [17], [23], [25], γ close to 1 (e.g., $\gamma > 0.99$) also causes the training instability problem. Thus, the analysis of the instability of undiscounted RL helps to alleviate the instability of optimizing a discounted return with a large γ.

As for analyzing this instability problem, the research in RL literature is limited to experimental studies [17], [19], [26], [27]. These studies show that the discount factor acts as a low-pass filter [17], [27], i.e., the bigger γ is, the weaker the ability is for reducing the noise in the learning process. Although these empirical results facilitate the adjustment of the discount factor, they lack a theoretical analysis of the cause of training instability and avoid optimization of undiscounted returns. As far as we know, the theoretical analysis of the instability problem only lies in classic dynamic programming (DP) literature [22]. Andrey Kolobov *et al.* [22] show that value iteration under undiscounted return will cause the policy to be trapped in sets of states with zero rewards, called *transient traps*. The limitation is that their analyses in DP are based on model-based/planning algorithms, and their method needs to traverse the transition model for the traps to be detected. However, modern RL algorithms are mostly data-based.

This article is the first to discover that transient trap also exists in model-free RL fields and provides theoretical analysis by combining the empirical results in RL and the theoretical analyzes in DP. Different from the insight of planning in DP, we analyze *transient trap* from the perspective of sampling. To illustrate the formation of transient traps, we present an example under Monte Carlo estimation [20], as shown in Fig. 2. The formation consists of the following two stages.

1) *Inconsistently selected actions* cause an *overestimation* of the nonoptimal actions. As shown in Fig. 2(a), the optimal action in state s is a_2, and $Q^\pi(s, a_2) = 1$. In this example, the optimal action a_2 is a consistent action for the estimation of $Q(s, a_1)$. The nonoptimal a_1 is a recurrent action, i.e., when the agent starts from s and explores a_1 first, the agent will revisit (RV) state s. After an RV, the agent may sample a_2 at this time and reach the goal state. It can be seen that $Q^\pi(s, a_1)$ will be close to $Q^\pi(s, a_2)$, and the inconsistency in the action sampling at s causes $Q^\pi(s, a_1)$ to be overestimated.
2) The overestimation causes *transient traps*. As shown in Fig. 2(b), when the value of nonoptimal action $Q^\pi(s, a_1)$ and the optimal action $Q^\pi(s, a_2)$ becomes close enough, the noise and randomness in the training process can make $Q^\pi(s, a_1)$ larger than $Q^\pi(s, a_2)$. In this situation, the agent following a greedy policy will always choose a_1 and RV state s at test time, forming a trap. The trap is transient because the update would penalize the trap trajectory and the suboptimal action. As a result, a transient trap occurs and disappears during training, leading to training instability.

Therefore, the inconsistency in action sampling needs to be corrected to eliminate the transient traps and stabilize the training process. An intuitive solution for eliminating the transient trap is to only select one consistent action in the same state. However, this sampling method limits exploration for other actions in episode [20]; algorithms with inefficient exploration are commonly not learnable. For balancing learnability and training stability, we propose to keep *partial actions* consistently selected, while the other actions can be inconsistently selected. This is reasonable because eliminating transient traps only requires a safe margin between the values of the optimal action and the nonoptimal actions. Based on this, this article proposes to decompose the undiscounted return into two parts, i.e., *last-visit (LV) return* and the *RV return*. The values of these two parts are evaluated separately, and they are added to replace the original value function. As shown in Fig. 3, the LV return only sums the part of the return after the LV to a state in each episode, while the RV return sums the remaining return in the episode. The separated evaluation of the return ensures that the LV return part will not be affected by inconsistently selected actions because the same state is not visited again afterward in the episode, thus correcting the overall estimation. Besides, as it will be proved in Section V, this decomposition does not change the optimal policy of the original task. For brevity, the method is referred to as LV sampling (LVS) in the rest of the article. LVS is first implemented with a dual-output neural network. Then, it is tested in various tasks including a representative grid-world task, a 3-D ViZDoom game [28], and several challenging robotic tasks based on the CoppeliaSim simulator [29]. Compared with time-awareness and vanilla baselines, our method exhibits superiority in sample efficiency and training stability. This shows that our method is efficient and effective in resolving the training instability problem.

The rest of this article is organized as follows. In Section II, the related literature is reviewed, and the drawbacks of major existing studies are discussed. In Section III, the preliminaries of our work are introduced. In Section IV, the transient trap problem is defined, and the cause of this problem is analyzed. In Section V, the LVS method is proposed to eliminate transient traps through partial consistency. Also, it is proven that this method can preserve the optimality of the policy, and the practical implementations of this method are presented.

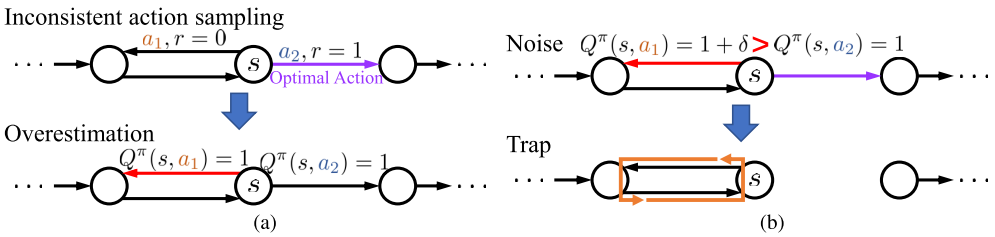

Fig. 2. Two stages of the formation of transient traps. (a) Stage 1: overestimation. (b) Stage 2: transient trap.

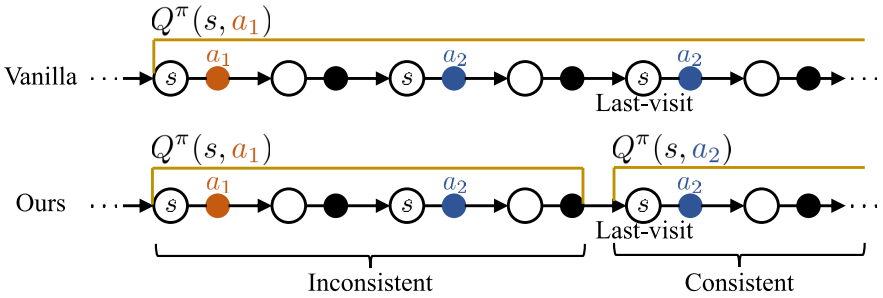

Fig. 3. Partial consistency produced by LVS.

In Section VI, the experimental results are provided to demonstrate the existence of transient traps and the effectiveness of the proposed LVS method. Section VII discusses the impact of this work and future research directions. The contributions are summarized as follows.

1) This article is the first to discover that the transient trap problem also exists in model-free undiscounted RL tasks, and it is a cause of the training instability problem.
2) This article performs a theoretical analysis of the formation mechanism of the transient trap problem. The analysis shows that the existence condition of transient traps is an overestimation of nonoptimal action caused by inconsistent action sampling.
3) This article proposes LVS, a new method for solving the transient trap problem by partially consistent action sampling. The effectiveness of the proposed method is demonstrated theoretically and empirically.

II. Related Work

The undiscounted return has been used as the testing criteria in many application fields [2], [5], [16], [30], [31]. Among them, two criteria are commonly used. One is "sum of rewards" (total reward [17], [31]); the other is "success probability of goal-reaching" (MAXPROB [12], [21]), e.g., cumulative scores [2], [30] in video games and success rates of robot tasks [5] [32]. However, undiscounted return is rarely taken as the learning criterion because it is difficult to optimize and can make algorithms fail to converge [13], [17].

In many famous algorithms, the undiscounted MDP is regarded as the "true" MDP, and the discount factor is used as a variance reduction technique [17], [24], [25]. However, the discount factor causes an inherent bias for undiscounted objective, which cannot be eliminated by designing value estimation, e.g., on-policy value estimation to address the bias in policy evaluation caused by exploration [33]. It does converge to the "best" policy that is determined by the discount factor. However, if the discount factor is inappropriately set, the converged policy may be nonoptimal for MDPs. Besides, when used for complex tasks, this technique will suffer from two problems: 1) tweaking or learning a suitable γ needs many trials and 2) optimizing the discounted return with a large γ ($\gamma > 0.99$) causes training instability [13], [17], [25]. Some studies exploit the temporal difference with a fixed horizon to stabilize RL [19], [20]. However, the representation of a long horizon or indefinite horizon by setting the fixed [20] needs task-specific designs of rewards or to learn to tweak the reward function [17].

Some studies optimized the undiscounted return, but they did not consider the "transient trap" problem. Salimans *et al.* [14] and Mania *et al.* [15] used evolution algorithms to learn policies. However, these algorithms require a complete evaluation before each update of policies, so they fail to achieve a high sampling efficiency. Cao *et al.* [16] and Pardo *et al.* [23] directly optimized the undiscounted return by augmenting the state space with the remaining time. Then, acyclic MDPs were constructed to eliminate transient traps. Although this method is task-independent, it enlarges the state space, which reduces the sampling efficiency in the training process [16]. Different from these methods, the method proposed in this article does not enlarge the search space and is more sampling efficient. R-learning [34] is a good method for solving tasks with an infinite sum of rewards. However, the tasks considered in this study are assumed to have a bounded total sum of rewards which are claimed in Section III. Thus, the core part of R-learning, i.e., the $\langle \rho, \sigma \rangle$ representation, does not work because ρ always converges to zero. In other words, R-learning is a method for representing an infinite sum of rewards. It does not modify the evaluation so that the overestimation of nonoptimal action remains in the MDPs, which contains finite total rewards and allows the existence of transient traps.

The "transient trap" problem has been investigated by the studies of the DP [35]–[38] literature. Especially, in generalized stochastic shortest path (GSSP) [22], the find, revise,

eliminate trap (FRET) algorithm was adopted to eliminate the transient trap. For RL problems, one core limitation of these works in DP is that they need to establish the transition model of the environment to detect the traps. Most modern RL algorithms are mostly data-based, and this characteristic is extended to data-based algorithms. It is revealed that the transient trap is the cause of the instability problem in undiscounted RL. From this perspective, our work has made the problem analysis in undiscounted returns more complete.

III. Preliminaries

In this section, the setting of undiscounted RL is introduced. An undiscounted MDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$, where $\mathcal{S}$ is the state space; $\mathcal{A}$ is the action space; $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function; and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function. We use $\mathcal{S}_g$ to denote the set of terminal states.

We consider that the horizon is infinite and assume that the total rewards are bounded. Since the return is bounded, undiscounted returns can be used without setting the average rewards [34]. Then, the objective of the undiscounted MDP is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected undiscounted return $U^\pi = \mathbb{E}^\pi[\sum_{t=0}^{\infty} R_t]$. The above settings of bounded rewards together with infinite horizon describe an MDP with sparse reward, which can characterize many realistic and complex problems for scaling up RL literature [17], [32]. In this setting, the undiscounted return has an obvious advantage in back-transporting long-term rewards.

IV. Transient Trap Problem

In this section, this article first describes the transient trap problem and analyzes how it causes training instability, i.e., the performance of policy fluctuates between suboptimality and optimality. Then, the existence condition of transient traps is derived. The theoretical analysis is based on the Monte Carlo estimation [20].

A. Definitions About Transient Trap

As mentioned in Section I, the implication of transient trap is two folds: 1) "trap" indicates that, during testing, the agent with the greedy policy will be trapped in a group of states and 2) "transient" indicates that, as training goes on, the trap will disappear because the update penalizes the trap trajectory. Also, the suboptimal action will lead to the trap since it will terminate before higher total rewards are collected. In summary, when the policy is tested in different episodes as training proceeds, the transient trap disappears and reappears, leading to performance fluctuations and nonconvergence of the training.

Since the trap is related to the topology structure of the transition model, the concept of "policy graph" is exploited to describe transient traps. For convenience, the notations from the DP literature [22] are adopted. The term "policy graph" is introduced as follows.

Definition 1 (Policy Graph): A *policy graph* $G^\pi = \{\mathcal{S}^\pi, \mathcal{E}^\pi\}$ is a directed graph that contains all the states reachable by the policy π, where $\mathcal{S}^\pi$ is the set of nodes (states) reachable by policy π, and $\mathcal{E}^\pi$ is the set of edges $\mathcal{E}^\pi = \{(s_i, s_j) | s_i, s_j \in \mathcal{S}^\pi, p(s_j|s_i, \pi(s_i)) > 0\}$.

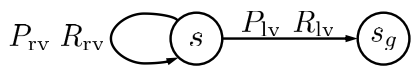

Fig. 4. Two-state abstracted MDP.

Accordingly, a *greedy graph* is the policy graph G^{π^Q} of a policy that always selects the action with the maximum value, i.e., $\pi^Q(s, a) = \mathrm{argmax}_{a \in \mathcal{A}} Q^\pi(s, a)$.

Based on this, a formal definition of the transient trap is given. As mentioned above, the transient trap characterizes: 1) a greedy graph consists of groups of communicating states but not connected to the outside and 2) the agent can move out of these groups of states by changing its policy. Thus, the "transient trap" is defined as follows.

Definition 2 (Transient Trap): A *trap* in a policy graph is a maximal strongly connected component (SCC), where there are no outgoing edges: $C = \{\mathcal{S}_C, \mathcal{E}_C\} \subsetneq G^\pi$, s.t. $\mathcal{S}_C \cap \mathcal{S}_g = \varnothing$ and $\forall s_i \in \mathcal{S}_C, s_j \in \mathcal{S} \setminus \mathcal{S}_C, (s_i, s_j) \notin \mathcal{E}^\pi$. A transient trap indicates that the agent can move out of the trap by changing its policy: $\exists s_i \in \mathcal{S}_C, s_j \in \mathcal{S} \setminus \mathcal{S}_C, a \in \mathcal{A}$, s.t. $p(s_j|s_i, a) > 0$.

Based on the definitions of policy graph and transient trap, the training instability problem caused by transient traps can be clearly described. Specifically, the *instability problem* is reflected by the policy graph oscillating between the optimal policy graph $G^{\pi^{Q^*}}$ and the nonoptimal policy graph G^{π^Q} containing *transient traps*.

B. Existence Condition of Transient Trap

In this section, this article analyzes the two stages in trap formation under the Monte Carlo estimation. In one stage, the inconsistently selected actions cause overestimation; in the other stage, the overestimation causes transient traps.

Stage 1 (Inconsistently Selected Actions Cause Overestimation): To find the part of the state-action value that causes overestimation of a nonoptimal state-action pair, the state-action value is decomposed by a two-state abstraction method proposed by Singh and Sutton [39].

This method abstracts the whole state space into state s (that is to be analyzed) and terminal state s_g, and it considers that the policy π at other states in the original MDP to be fixed. The trajectories of the agent involve several visits to s and the final termination at s_g. The abstracted MDP is shown in Fig. 4, where R_{rv} is the reward obtained between RVs s; R_{lv} is the reward obtained between the LV to s and the termination; and P_{rv} and P_{lv} are the abstracted transition probabilities.

In this way, the state-action value is decomposed according to the transitions to terminal states and those to an RV. Then, the following lemma can be formulated.

Lemma 1 (Value Function Decomposition): Let "lv" denote the LV of s and "rv, k" denote the kth RV to s. Also, note that $V_{\mathrm{rv,k}}(s) = V(s)$ due to infinite horizon. Then, for each state s, the value function satisfies

$$Q^\pi(s, a) = P_{\mathrm{lv}} R_{\mathrm{lv}} + P_{\mathrm{rv}}[R_{\mathrm{rv}} + V^\pi(s)].$$

Proof: The proof is provided in the Supplementary Material. □

The above decomposition shows that, for $s \in \mathcal{S}$ and $a \in \mathcal{A}$, the state-action value $Q^{\pi}(s_t, a_t)$ depends on the state values of its RVs $V^{\pi}(s)$, as shown in Fig. 4.

During training, when different actions (e.g., a greedy action and an exploration action) are selected at different visits in the same state, the values of former visits would be changed by mistake. For a nonoptimal action, this mistake will cause the optimal action value to be added to the nonoptimal action's value, resulting in the overestimation of the nonoptimal action. Although, in discounted RL, this mistake can be eliminated by γ, it is still prominent in undiscounted RL. The above analysis indicates that, for a nonoptimal action, the probability of P_{rv} leading to an RV determines the severity of overestimation. The higher the probability, the more severe the overestimation, and the less distinguishable the nonoptimal and optimal action values. As will be introduced in Section V, when P_{rv} becomes large, the nonoptimal action value can become close to the optimal action value. In this case, the optimal policy graph is fragile, and some noises can cause the optimal action to be replaced with the nonoptimal action, forming a transient trap.

Stage 2 (Overestimation Causes Transient Trap): Here, the formation of a transient trap by overestimation is analyzed. According to its definition, a transient trap is an SCC without outgoing edges, indicating that $P_{\text{rv}} = 1$. However, $P_{\text{rv}} = 1$ does not strictly hold in practice due to the stochastic policy and the transition model. For clarity, the formation of a transient trap is first analyzed based on $P_{\text{rv}} = 1$ under deterministic MDP. Then, in the rest part, the assumption of $P_{\text{rv}} = 1$ is relaxed to show the suboptimality in general cases with $P_{\text{rv}} < 1$.

Here, it will be proved that the optimal action will be replaced by a nonoptimal action with $P_{\text{rv}}(s, a) = 1$, leading to a transient trap. This is because the nonoptimal action and the optimal action tend to have the same values. Thus, a little noise in the learning process can make the optimal action be replaced with a nonoptimal one. As a result, the agent will always choose the recurrent actions during testing and, thus, be trapped in a group of states. Based on Lemma 1, we analyze the policy on state s and fix the policy on the other states $\hat{s}$ to be greedy, which is necessary for $P_{\text{rv}}(s, a) = 1$. The analysis result is summarized in the following theorem.

Theorem 1 (Transient Trap in a Deterministic MDP): Given an optimal Q-function Q^*, let the state to be considered be s and its optimal action be a^*. We only consider the policy update on s and fix the policy on the other states $\hat{s}$ to be greedy, i.e., $\pi^{Q^*}(\hat{s})$. Also, an arbitrarily small noise δ in the training is considered. Then, a transient trap will be formed in the subsequent policy graph *if and only if* there exists a nonoptimal action $a \neq a^*$ in the current greedy graph of Q^* such that $P_{\text{rv}}(s, a) = 1$.

Proof: The proof is provided in the Supplementary Material. □

The above theorem reveals that the optimal action will be replaced by a nonoptimal action that causes a transient trap. Then, we give a typical example to describe the cases in which nonoptimal action with $P_{\text{rv}} = 1$ can exist.

Example 1 (Zero-Reward Cycles): As shown in Fig. 2(a), the zero-reward cycle is an SCC consisting of two states with no rewards inside. The state-action transition probabilities in the two states are deterministic. This SCC can have outgoing edges under appropriate policies, but the agent can change its policy to make the SCC closed. It should be noted that most MDPs with sparse rewards contain zero-reward cycles [22].

Then, we release the assumption of $P_{\text{rv}} = 1$ because, in practice, the policy is commonly stochastic. However, the overestimation of nonoptimal actions still commonly exists. This overestimation can still cause the values of nonoptimal actions to be larger than those of the optimal due to the noise in the learning process. Formally, the estimated state-action value is denoted as $\hat{Q}(s, a) = Q(s, a) + \delta$, where $Q(s, a)$ is the expected value and δ is the noise in value estimation during training, including the noises caused by sample variance, learning rate, and computation accuracy. The analysis is performed also based on Lemma 1: we analyze the policy on state s and fix the policy on other states to be ϵ-greedy policy. In this case, $P_{\text{rv}} < 1$ commonly holds. Then, we provide the condition for the overestimation to cause a nonoptimal action to replace the optimal action on s.

Theorem 2 (Training Instability Under the Value Noise): Given an optimal Q-function Q^*, let the state considered be s and its optimal action be a^*, and a nonoptimal action be a. We only consider the policy update on s and fix the policy on the other states $\hat{s}$ to be ϵ-greedy. Then, the following two propositions are equivalent.

1) During testing, the greedy policy $\pi^{Q}(s)$ is suboptimal.
2) During training, one of the following two cases occurs:

$$P_{\text{rv}}(s, a) > \frac{Q^*(s, a^*) - R_{\text{lv}} - \delta}{V^*(s) + R_{\text{rv}} - R_{\text{lv}}}$$

when $V^*(s) + R_{\text{rv}} - R_{\text{lv}} > 0$, $Q^*(s, a^*) - R_{\text{lv}} > 0$, or

$$P_{\text{rv}}(s, a) < \frac{Q^*(s, a^*) - R_{\text{lv}} - \delta}{V^*(s) + R_{\text{rv}} - R_{\text{lv}}}$$

when $V^*(s) + R_{\text{rv}} - R_{\text{lv}} < 0$ and $Q^*(s, a^*) - R_{\text{rv}} - V^*(s) > 0$.

Proof: The proof is provided in the Supplementary Material. □

The above theorem reveals that, in practice, $P_{\text{rv}} = 1$ is not a necessity to achieve suboptimality, and the actual condition is weaker

$$P_{\text{rv}}(s, a) > \frac{Q^*(s, a^*) - R_{\text{lv}} - \delta}{V^*(s) + R_{\text{rv}} - R_{\text{lv}}}.$$

When $\gamma < 1$, $V^*(s)$ can be decayed, and thus, the right term in the above inequality is large. In this case, the learning process is resistant to relatively large noise. Meanwhile, the smaller γ is, the larger noise the training process can withstand. When $\gamma = 1$, $V^*(s)$ cannot be decayed. Note that $Q^*(s, a^*)$ and $V^*(s)$ get close because $\epsilon-$greedy is a soft policy about π^{Q^*}, i.e., $V^*(s) = \epsilon((\sum Q(s, \hat{a}))/N) + (1 - \epsilon)Q(s, a^*)$, where N is the number of nonoptimal actions $\hat{a}$. In this case, a relatively small noise can make

$$\frac{Q^*(s, a^*) - R_{\text{lv}} - \delta}{V^*(s) + R_{\text{rv}} - R_{\text{lv}}} < 1$$

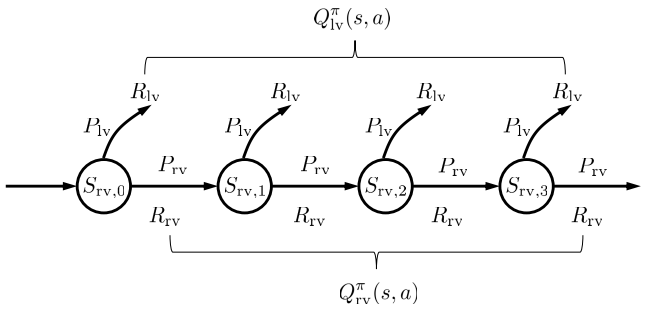

Fig. 5. Illustration of P_{lv} and P_{rv}. Circles indicate different visits to the state s in a trajectory.

such that, if the P_{rv} is larger than the above value, nonoptimal action will replace the optimal action.

Besides, the second case in Theorem 2 indicates that, under value noises, there exists another situation that the suboptimal action replaces the optimal one due to inconsistent action sampling, that is, when R_{lv} is large and P_{rv} is small. Interestingly, this case has less impact on the optimal value than the first one because the difference between optimal and suboptimal state-action values is $P_{\mathrm{rv}}(V^*(s)-V^{\pi}(s))$, as shown in (1). The degree of impact depends on the probability of RVs P_{rv}. This theorem identifies more scenarios leading to training instability than transient traps, which has made the problem analysis in the training instability of undiscounted return more complete.

This analysis indicates that the inconsistently selected actions cause the overestimation of nonoptimal action value first; then, the overestimation causes the values of nonoptimal actions to be larger than those of the optimal action when the policy deduces a large $P_{\mathrm{rv}}(s,a)$, and suboptimality is caused. In the most serious cases, $P_{\mathrm{rv}}(s,a)=1$ causes a transient trap. Thus, the action inconsistency needs to be alleviated to eliminate the transient traps so that the training process can be stabilized. In Section V, a method will be proposed to solve this problem.

V. Partial Consistency for Eliminating Transient Traps

In Section IV, the analysis has shown that transient traps are formed by inconsistently selected actions. In this section, for eliminating transient traps, a novel sampling method is proposed to eliminate transient traps and keep partial action selections consistent.

A. Method

As the inconsistency needs to be addressed to eliminate transient traps, the selected actions should be consistent. For deterministic policies, consistent action selecting indicates that the agent can only explore one action in the same state of each episode. That is, if the agent selects an action for exploration in a state, it should select this action whenever it returns to this state; otherwise, all the samples with inconsistently selected actions should be discarded. However, it is emphasized by Sutton and Barto [20] that keeping all selected actions consistent in episodes is severely inefficient for exploration and sampling.

Based on this analysis, this article proposes to achieve "*partial consistency*" instead of selecting consistent actions. The "*partial consistency*" implies that consistency is ensured in a subset of samples. This is reasonable because alleviating overestimation only requires $Q^{\pi}(s_t,a_t)$ to be not close to $Q^{\pi}(s_t,a_t^*)$ when $P_{\mathrm{rv}}(s_t,a_t)$ is large, instead of an accurate estimation of $Q^{\pi}(s_t,a_t)$. "*Partial consistency*" can provide a safe margin between $Q^{\pi}(s_t,a_t)$ and $Q^{\pi}(s_t,a_t^*)$. However, in this case, the return will be changed and different from the commonly used first-visit return or every-visit return [39], which may lead to suboptimal policies. Therefore, our proposed method needs to preserve the unbiased estimation of the original state-action values. Two conditions for effective selection of the subset of samples are summarized.

1) *Effectiveness for Eliminating Traps:* The subset should not contain rewards between RVs; otherwise, it will count the returns from inconsistently selected actions after revisiting a state.
2) *Optimality Preservation:* The subset should ensure that the rest part of the return can be transformed into a conditional expectation. That is, when a return is separately evaluated under some conditions, what we evaluate is a conditional expected return.

Under the first condition, the part of the return that has followed the LV to the state is chosen for separate sampling and evaluation because there is no RV after the last time of visiting a state in a trial. Accordingly, we call it LV return, i.e., $Q^{\pi}_{\mathrm{lv}}(s,a)=\mathbb{E}^{\pi}[\sum_{i=t_{\mathrm{lv}}}^{\infty}R_i]$, where t_{lv} is the time of LV. After the LV return is deducted from the total expected return, the remaining part is the average of the returns between (re)visits to the state, and it is called RV return, i.e., $Q^{\pi}_{\mathrm{rv}}(s,a)=\mathbb{E}^{\pi}[\sum_{i=t}^{t_{\mathrm{lv}}-1}R_i]$. In this article, this sampling method is referred to as LVS. Similar to (1), the state-action value function in "k, rv" of LVS is expressed as

$$Q^{\pi}_{\mathrm{rv},k}(s,a) = P_{\mathrm{lv}}R_{\mathrm{lv}}+P_{\mathrm{rv}}\left[R_{\mathrm{rv}}+\underbrace{Q^{\pi}_{\mathrm{lv}}(s,a)}_{\text{Partial consistency}}+\left[\epsilon\frac{\sum Q^{\pi}_{\mathrm{rv},k+1}(s,\hat{a})}{N}+(1-\epsilon)Q^{\pi}_{\mathrm{rv},k+1}(s,a^*)\right]\right].$$

LVS constrains that the LV return in "rv, $k+1$" must be generated by the consistency action (without exploring other actions), while the other part of the return can be generated by inconsistently selected actions.

Under the second condition, a conditional expectation is evaluated, which is called LV conditioned expected return, i.e., $Q^{\pi}_{\mathrm{lv}}(s,a)=\mathbb{E}^{\pi}[\sum_{i=t_{\mathrm{lv}}}^{\infty}R_i|s \text{ is lv}]$. However, the unweighted LV conditioned expected return and RV return may not be summed directly, probably leading to a biased evaluation of the

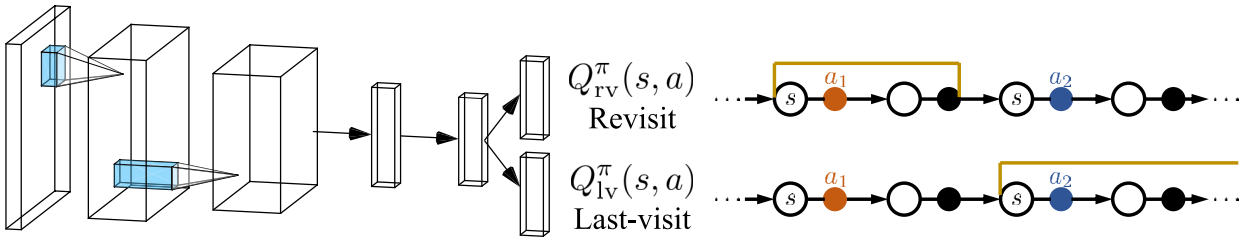

Fig. 6. Dual-output network structure and division of samples for implementing LVS.

optimal policy. Fortunately, it is noted that the prior probability of last-visiting a state might be equal to 1 because any trial has a termination, and the same states must be last-visited once. Formally, an expected return can be divided into two parts, i.e., RV expected return and LV conditioned expected return.

Theorem 3 (Optimality of LVS): For deterministic policies, the state-action value function $Q^{\pi}(s,a)$ can be transformed to the sum of the RV return and the LV conditioned return

$$Q^{\pi}(s,a) = \underbrace{\mathbb{E}^{\pi}\left[\sum_{k=t}^{t_{\mathrm{lv}}-1} R_k | s_t = s, a\right]}_{\text{revisit}} + \underbrace{\mathbb{E}^{\pi}\left[\sum_{i=t_{\mathrm{lv}}}^{\infty} R_i | s \text{ is lv}, a\right]}_{\text{last-visit conditioned}}.$$

Proof: As shown in Fig. 5, $Q^{\pi}(s,a)$ can be divided according to the RV times. By expanding it in regard to the number of RVs before the LV, "rv, k," we have

$$\begin{aligned}
&\mathbb{E}^{\pi}\left[\sum_{i=t}^{\infty} R_i\right] \\
&= \left[\sum_{\tau_{\mathrm{lv}}\in\Gamma_{\mathrm{lv}}} p(\tau_{\mathrm{lv}}|\mathrm{rv},0) + \sum_{\tau_{\mathrm{rv}}\in\Gamma_{\mathrm{rv}}} p(\tau_{\mathrm{rv}}|\mathrm{rv},0)\right]\sum_{i=t_{\mathrm{rv},0}}^{\infty} R_i \\
&\quad + P_{\mathrm{rv},1}\left[\sum_{\tau_{\mathrm{lv}}\in\Gamma_{\mathrm{lv}}} p(\tau_{\mathrm{lv}}|\mathrm{rv},1) + \sum_{\tau_{\mathrm{rv}}\in\Gamma_{\mathrm{rv}}} p(\tau_{\mathrm{rv}}|\mathrm{rv},1)\right]\sum_{i=t_{\mathrm{rv},1}}^{\infty} R_i \\
&\quad + P_{\mathrm{rv},1}P_{\mathrm{rv},2}\left[\sum_{\tau_{\mathrm{lv}}\in\Gamma_{\mathrm{lv}}} p(\tau_{\mathrm{lv}}|\mathrm{rv},2) + \sum_{\tau_{\mathrm{rv}}\in\Gamma_{\mathrm{rv}}} p(\tau_{\mathrm{rv}}|\mathrm{rv},2)\right]\sum_{i=t_{\mathrm{rv},2}}^{\infty} R_i \\
&\quad + \cdots
\end{aligned}$$

where Γ_{lv} is the set of trajectories without RVs to s and Γ_{rv} is the set of trajectories that will RV s. At every RV, we consider the term $\sum_{\tau_{\mathrm{lv}}\in\Gamma_{\mathrm{lv}}} p(\tau_{\mathrm{lv}}|\mathrm{rv},k)\sum_{i=t_{\mathrm{rv},k}}^{\infty} R_i$. With the Markov property and infinite-horizon assumption, we have the following results.

1) $P_{\mathrm{rv},1}(s) = P_{\mathrm{rv},2}(s) = P_{\mathrm{rv},3}(s) = \cdots$
2) $\sum_{\tau\in\Gamma_{\mathrm{lv}}} p(\tau|\mathrm{rv},0)\sum_{i=t_{\mathrm{rv},0}}^{\infty} R_i = \sum_{\tau\in\Gamma_{\mathrm{lv}}} p(\tau|\mathrm{rv},1)\sum_{i=t_{\mathrm{rv},1}}^{\infty} R_i = \cdots$

Therefore, we have the following results:

$$\mathbb{E}^{\pi}\left[\sum_{i=t_{\mathrm{lv}}}^{\infty} R_i\right] = \left(1 + P_{\mathrm{rv}} + P_{\mathrm{rv}}^2 + \cdots\right)\sum_{\tau\in\Gamma_{\mathrm{lv}}} p(\tau|\mathrm{lv})\sum_{i=t_{\mathrm{lv}}}^{\infty} R_i.$$

Then, we are going to show that the LV part of the return can be estimated under the condition that s is last visited. We make a transformation of the last two-terms from simple summation to the conditional expectation

$$\begin{aligned}
\sum_{\tau\in\Gamma_{\mathrm{lv}}} p(\tau|\mathrm{lv})\sum_{\mathrm{i}=\mathrm{t_{lv}}}^{\infty}\mathrm{R_i} &= P(\tau\in\Gamma_{\mathrm{lv}})\sum_{\tau\in\Gamma_{\mathrm{lv}}}\frac{p(\tau|\mathrm{lv})\sum_{\mathrm{i}=\mathrm{t_{lv}}}^{\infty}\mathrm{R_i}}{P(\tau\in\Gamma_{\mathrm{lv}})} \\
&= [1 - P_{\mathrm{rv}}(s)]\mathbb{E}\left[\sum_{i=t_{\mathrm{lv}}}^{\infty} R_i | s \text{ is lv}\right].
\end{aligned}$$

Accordingly, we have

$$\begin{aligned}
\mathbb{E}^{\pi}\left[\sum_{i=t_{\mathrm{lv}}}^{\infty} R_i\right] &= [1 - P_{\mathrm{rv}}(s)]\left[1 + P_{\mathrm{rv}} + P_{\mathrm{rv}}^2 + \cdots\right] \\
&\quad \times \mathbb{E}\left[\sum_{i=t_{\mathrm{lv}}}^{\infty} R_i | s \text{ is lv}\right].
\end{aligned}$$

Note that $(1 - P_{\mathrm{rv}})(1 + P_{\mathrm{rv}} + P_{\mathrm{rv}}^2 + \cdots) = 1$ due to the long-horizon assumption of the episodes. Finally, we find that

$$\begin{aligned}
\mathbb{E}^{\pi}\left[\sum_{i=t}^{t_{\mathrm{lv}}} R_i\right] &= \sum_{\tau_{\mathrm{rv}}\in\Gamma_{\mathrm{rv}}} p(\tau_{\mathrm{rv}}|\mathrm{rv},0)\sum_{i=t_{\mathrm{rv},0}}^{\infty} R_i \\
&\quad + P_{\mathrm{rv},1}\sum_{\tau_{\mathrm{rv}}\in\Gamma_{\mathrm{rv}}} p(\tau_{\mathrm{rv}}|\mathrm{rv},1)\sum_{i=t_{\mathrm{rv},1}}^{\infty} R_i \\
&\quad + P_{\mathrm{rv},1}P_{\mathrm{rv},2}\sum_{\tau_{\mathrm{rv}}\in\Gamma_{\mathrm{rv}}} p(\tau_{\mathrm{rv}}|\mathrm{rv},2)]\sum_{i=t_{\mathrm{rv},2}}^{\infty} R_i + \cdots
\end{aligned}$$

define the expected return between t and t_{lv}. Therefore, the Q-function can be divided as shown in the following equation:

$$Q^{\pi}(s,a) = \underbrace{\mathbb{E}^{\pi}\left[\sum_{k=t}^{t_{\mathrm{lv}}-1} R_k | s_t = s, a\right]}_{\text{revisit}} + \underbrace{\mathbb{E}^{\pi}\left[\sum_{i=t_{\mathrm{lv}}}^{\infty} R_i | s \text{ is lv}, a\right]}_{\text{last-visit conditioned}}$$

which completes the proof. □

The above theorem reveals that the original value function is preserved in LVS. Based on this equivalence, the value of $Q^{\pi}(s,a)$ can be calculated by directly summing the unweighted RV return and the LV conditioned return. It should be noted that, in this section, we do not assume the environment to be stochastic or deterministic, i.e., the above theorem holds in both stochastic and deterministic environments.

Next, the elimination of transient traps by LVS is described. The separated evaluation of $Q^{\pi}_{\mathrm{lv}}(s,a)$ and $Q^{\pi}_{\mathrm{rv}}(s,a)$ implies that, within the same trajectory, different parts of the trajectories are used to estimate different state-action values. The LV part selects states that are last-visited from the trajectories and then counts the returns that have followed these states. The RV part counts the returns of the other states with the first or every-visit estimation. Since the part $Q^{\pi}_{\mathrm{lv}}(s,a)$ is obtained by

consistently selected actions, the case in which the nonoptimal action value is too close to the optimal action value can be avoided to eliminate the transient trap. As for the effectiveness of our method, a theorem is presented under the most severe case of $P_{\text{rv}}(s,a)=1$.

Theorem 4 (Effectiveness of LVS for Eliminating Traps): For the greedy graph G^{Q^*} of an optimal Q-function Q^*, under LVS, there is no transient trap even if there exists a state-action pair (s,a) s.t. $P_{\text{rv}}(s,a)=1$.

Proof: The proof is provided in the Supplementary Material. □

Our theorem reveals that, if there is already an optimal policy, unlike first-visit return or every-visit return, LVS will not generate transient traps in the policy graph. Thus, LVS should be an effective improvement to eliminate traps and improve training stability in undiscounted RL. Note that LVS does not penalize trajectories with transient traps, which may introduce a bias to the value estimation. Instead, LVS eliminates transient traps by enhancing the consistency in action sampling. Because an action that causes transient traps is suboptimal, its value is lower than the value of an optimal action under consistent action sampling. Thus, the enhancement of action sampling consistency provided by LVS offers a safe margin, also called action gap [40], between the values of nonoptimal actions and the values of optimal actions.

B. Implementation With Function Approximation

In this section, the implementation of our method with parameterized function approximators is described in detail. The neural networks used to implement LVS and the LV return, as well as the RV return, are illustrated in Fig. 6. LVS requires an estimation method that estimates the LV conditioned return separately. To this end, Monte Carlo is selected as the estimation method. Other mainstream estimation methods, such as temporal difference, are not selected by this study because they use bootstrapping and may count the returns of greedy actions on that of nongreedy actions after revisiting a state.

1) Monte Carlo With LVS: For each state in a roll-out, the transitions of each state are divided into two parts: the RV part $\sum_{i=t}^{t_{\text{lv}}-1} r_i$ and the LV part $\sum_{i=t_{\text{lv}}}^{\infty} r_i$. First, the RV Q-function $Q_{\text{rv}}^{\xi}(s,a)$ parameterized by ξ is used to estimate the RV expected return, i.e., $\mathcal{L}_{\xi}(s,a) = ||Q_{\text{rv}}^{\xi}(s,a) - \sum_{i=t}^{t_{\text{lv}}-1} r_i||_2^2$. Then, the LV Q-function $Q_{\text{lv}}^{\theta}(s_{\text{lv}},a_{\text{lv}})$ parameterized by θ is used to estimate the LV conditional expected returns, i.e., $\mathcal{L}_{\theta}(s_{\text{lv}},a_{\text{lv}}) = ||Q_{\text{lv}}^{\theta}(s_{\text{lv}},a_{\text{lv}}) - \sum_{i=t_{\text{lv}}}^{\infty} r_i||_2^2$. Next, the two Q-functions are combined to obtain the overall Q-function

$$Q_{\xi,\theta}(s,a) = Q_{\text{rv}}^{\xi}(s,a) + Q_{\text{lv}}^{\theta}(s,a).$$

In the experiments introduced in Section VI, this algorithm is used in the tasks of Cliff-Walking, Gathering, and ViZDoom navigation. These tasks differ in the implementation of the value estimators. Specifically, a Q-table is used for the Cliff-Walking and the Gathering task, and a Q-network is used for the ViZDoom task.

Algorithm 1 LVS-Based Actor–Critic

Input: last-visit Q-function Q_{lv}, revisit Q function Q_{rv}, policy π, episode buffer D, last-visit buffer D_{lv}
for episode$=1$ **to** M **do**
  Reset buffer D and D_{lv}; get initial observation s_1
  for $t=1$ **to** T **do**
    Sample an action a from the distribution $\pi(a|s)$
    Execute a_t, get (s_t,a_t,r_t,s_{t+1}), and store in D
  end for
  for each transition (s_t,a_t,r_t,s_{t+1}) in D **do**
    if $\exists i \in \{t+1,\ldots,T\}$ s.t. $s_i=s_t$ **then**
      Copy (s_t,a_t,r_t,s_{t+1}) to D_{lv}
    end if
  end for
  for each transition (s_t,a_t,r_t,s_{t+1}) in D_{lv} **do**
    $y_{t,\text{lv}} \leftarrow \sum_{i=t}^{\infty} r_i$
  end for
  for each transition (s_t,a_t,r_t,s_{t+1}) in D **do**
    $y_{t,\text{rv}} \leftarrow \sum_{i=t}^{\infty} r_i$
    if $\exists (s_i,a_i,r_i,s_{i+1}) \in D_{\text{lv}}$ s.t. $s_t=s_i$ **then**
      $y_{t,\text{rv}} \leftarrow y_{t,\text{rv}} - y_{i,\text{lv}}$
    end if
  end for
  Update Q_{lv} with loss $\left\| y_{t,\text{lv}} - Q_{\text{lv}}(s_t,a_t) \right\|_2$
  Update Q_{rv} with loss $\left\| y_{t,\text{rv}} - Q_{\text{rv}}(s_t,a_t) \right\|_2$
  Update π with ac loss or PPO loss
end for

2) REINFORCE With LVS: Although the above theoretical analysis assumes a deterministic policy, the performance of LVS is evaluated under popular stochastic policies, including A3C [41] and proximal policy optimization (PPO) [42]. It should be noted that the critic part of the original PPO and A3C policies is implemented with advantage functions. Similar to the REINFORCE algorithm [20], the Monte Carlo estimation is adopted to implement the critic in the actor–critic framework in this work. For fairness, the Monte Carlo estimation is taken as the baseline. π^{ζ} parameterized by ζ is used to show the loss functions of REINFORCE-AC and REINFORCE-PPO

$$\mathcal{L}_{\text{ac}}(s,a) \propto -\mathbb{E}^{\pi}\left[Q_{\xi,\theta}(s,a)\log(\pi^{\zeta}(a|s))\right]$$

$$\mathcal{L}_{\text{PPO}}(s,a) \propto -\mathbb{E}^{\pi}\left[\min\left[Q_{\xi,\theta}(s,a)\frac{\pi^{\zeta}(a|s)}{\pi_{\text{old}}^{\zeta}(a|s)}, Q_{\xi,\theta}(s,a)\text{clip}\left\{\frac{\pi^{\zeta}(a|s)}{\pi_{\text{old}}^{\zeta}(a|s)}\right\}\right]\right]$$

where $Q_{\xi,\theta}(s,a)$ is estimated by Monte Carlo estimation, i.e., $Q_{\xi,\theta}(s,a) = Q_{\text{lv}}^{\theta}(s,a) + Q_{\text{rv}}^{\xi}(s,a)$. The estimations of $Q_{\text{lv}}^{\theta}(s,a)$ and $Q_{\text{rv}}^{\xi}(s,a)$ are the same as those of the above deterministic policies.

Accordingly, several algorithms are designed for stochastic policies within the actor–critic framework. As a representative algorithm, the algorithm of LVS actor–critic is presented in Algorithm 1.

3) Recognition of LV: One challenge to implement LVS is to measure the "state similarity" in MDPs with continuous

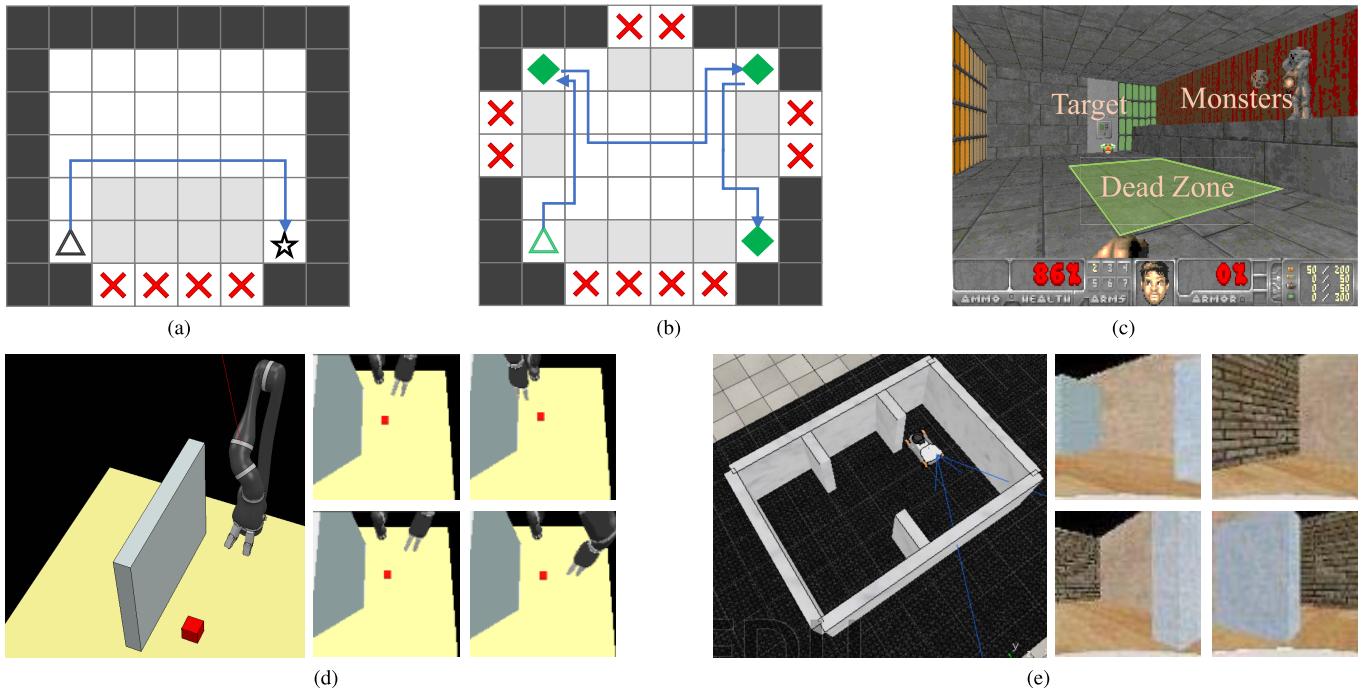

Fig. 7. Overview of the environments and the agent's observations in the five tasks. (a) Cliff-Walking: reach the star. (b) Gathering: collect the green diamonds. (c) ViZDoom navigation: navigate to the target place. (d) Jaco grasping: grasp the red object. (e) Youbot covering: cover the whole room. The gray shaded areas in (a) and (b) indicate randomness in the environments. Randomness in (c)–(e) is produced by simulators. (a) Cliff-Walking. (b) Gathering. (c) ViZDoom navigation. (d) Jaco grasping (left: overview; right: agent's observations). (e) Youbot covering (left: overview; right: agent's observations).

state spaces. Similar to curiosity-driven exploration [43], the self-supervised prediction [44], [45] or successor feature [46], [47] can be exploited to measure the state similarities in continuous cases in the future work. The analysis of the experimental results in this work just uses the position information provided by the simulator. Meanwhile, thresholds are set to perform the recognition. Details are in the experimental settings.

VI. Experiments

In this section, the experimental results are presented. The changes in training stability with a discount factor γ are shown first. Also, the transient traps that exist in the training process with $\gamma = 1$ are visualized. Then, the effectiveness of our LVS method in eliminating the transient trap problem and stabilizing the training process is demonstrated. Finally, it is shown that LVS is effective for both deterministic and stochastic policies.

A. Environmental Settings

Two types of tasks were considered in the experiment, i.e., the tasks with discrete state space and the tasks with continuous state space. The former is convenient for visualizing the agent's policy and trajectories, while the latter verifies the algorithms' capability to solve complex tasks.

The first type of task includes a Cliff-Walking task [as shown in Fig. 7(a)] and a Gathering task [as shown in Fig. 7(b)], which, respectively, represents the tasks with a single reward and multiple rewards. The second type of tasks includes a navigation task in a ViZDoom game environment [28] [as shown in Fig. 7(c)], a commonly used manipulator grasping task [48] in the CoppeliaSim simulator [as shown in Fig. 7(d)], and a robot floor-covering task [49] [as shown in Fig. 7(e)]. In the first type of task, the agent perceives its current position; in the second type of task, the agent perceives an RGB image of the environment. For all the above five tasks, the environmental rewards are set to be sparse, and the horizons are set to be much longer than required for an approximate expression of the infinite horizon. Note that, in these tasks, finding the shortest path is not required. Instead, every path that obtains the maximum total reward in a certain timeframe is acceptable. Specifically, the objectives of Cliff-Walking, VIZDoom navigation, and Jaco grasping are MAXPROB [34], where a positive reward is assigned to the goal state, zero rewards are assigned for other states, and an undiscounted form of return is adopted. The objectives of Gathering and YouBot covering are to maximize the total reward [17], where some positive rewards are assigned to a certain set of states, and zero rewards are assigned for other states. Note that step penalty is not included in the reward function for unbiased expression of the original task objective. To simulate the infinite horizon, the maximum length of episodes was set to be more than ten times, which is required by the trajectory of optimal policy. The details are provided as follows.

1) Cliff-Walking Task: The objective of the agent is to walk from the origin state marked with a triangle to the target state marked with a star. The area marked with red crosses is the cliff: if the agent moves to this area, the episode will terminate, and the agent receives a reward of 0. The "slip" gray-shaded area winds with a certain probability $p_{\text{slip}} = 0.4$, making the agent uncontrollably move randomly. As mentioned above,

we set a very long horizon to 100, which is about ten times the step number required by the optimal policy. The observation of the agent is the position tuple (x, y), and the action space of the agent is {move down, move up, move left, move right}. We set a binary sparse reward function as one for the target state and zero for the other states.

2) Gathering Task: The objective is to gather all the items marked with green diamonds. The areas marked with red crosses are the dead areas and gray-shaded areas are "slip" areas, of which the settings are the same as those in the Cliff-Walking task. We set the horizon to 150, which is about ten times the step number required by the optimal policy. The observation of the agent is the tuple $(x, y, \sum r)$, where $\sum r$ is the number of items that have been collected. Adding $\sum r$ maintains the Markovian property of this environment and avoids the partial observation problem. The action space of the agent is {move down, move up, move left, move right}. As for the setup of the reward function, we let the environment only offer the agent reward of 1 when the agent gathers a new green diamond with an item and 0 otherwise.

3) ViZDoom Navigation: The objective of the agent is to reach the "Target." The "Dead Zone" is an area in which the agent will be attacked by the monsters. Since the shooting of the monsters has randomness, the agent can be killed with some probability, which depends on the time of the agent staying in the area and the distance between the agent and the monsters. To represent the indefinite horizon in this task, we set the horizon to 300 which is about six times the step number required by the optimal policy. The observations of the agent are RGB inputs. The action space of the agent is {move forward, turn left, turn right}. We set binary sparse reward as one for moving to the target state and zero for the other states.

4) Jaco Grasping: The objective of the agent is to grasp the red square. The robot arm must avoid collision with the obstacle, i.e., the gray wall. We set the motor with a certain noise when the robot arm is in a straight line, making the end-effector uncontrollably move randomly with the probability $P(s, a_{\text{random}}) = 0.4$. To represent the indefinite horizon in this task, we set the horizon to 100, which is about ten times the step number of the optimal policy. The observation of the agent is an RGB image. The action space of the agent is {moving forward, moving back, moving left, moving right}. We set binary sparse reward as one for grasping the object and otherwise zero.

5) Youbot Covering: The objective of the agent is to cover as much area as possible without collision with the wall. To represent the indefinite horizon in this task, we set the horizon to 200, which is about ten times the step number required by the optimal policy. The observation of the agent is an RGB image. The action space of the agent is {turn left, turn right, move forward}. As for the setup of the reward function, we split the ground area into six grids to allocate reward signals: in each episode, once the agent reaches a new grid, it receives a $+1$ reward; once the agent collides with the wall, it receives a -0.1 reward. Also, we add $\sum r$ into the state channels to alleviate partial observation.

B. Algorithm Settings

To verify the effectiveness of LVS in data-based fields, model-free RL and model-based RL are considered. Under model-free RL, the agent learns its policy directly by estimating the value function on data sampled from the interaction with the environment. Under model-based RL, the agent first learns a model of the environment's dynamics (world model) on the sampled data and then learns the policy based on the world model. This article uses the popular dreamer framework [50] for the model-based RL. The dreamer framework learns a reward model, a function representation model, and a transition model. Thus, the policy can be updated in multiple steps only through interactions with the world model, which obviously accelerates the learning process. Model-free and model-based agents take Crude Monte Carlo, REINFORCE-PPO, and REINFORCE-A3C as the basic algorithms for policy learning. This has been introduced in Section V-B.

1) LVS—Measurement of State Similarity: As for the measurement of state similarity in the tasks where the state space is continuous, we use the position-angle tuple (x, y, α). (x, y) is the displacement relative to the initial position, and α is the rotation angle relative to the initial position. In the ViZDoom navigation tasks, we approximate (x, y) with a precision of 20 points of size and α with a precision of 20°. The approximations are used to compare whether two states are the same to determine the occurrence of RVs. In the Jaco grasping task, we use the position tuple (x, y), which is the displacement of the end-effector relative to the initial position. We approximate the position with a precision of 0.1 m and use the approximation to determine whether an RV happens: if and only if the approximation of a state is the same as the approximation of a previous state, we regard that an RV happens. In the Youbot Covering task, we use the position-angle tuple (x, y, α), where x, y is the displacement relative to the initial position. We approximate the position with a precision of 0.1 m and α with a precision of 10° and use the approximation to determine whether RVs happen.

2) Baselines: We selected two baselines for our experiment. The first one is the vanilla algorithm. It was selected to test whether our core design of LVS improves the training stability. The second one is the time-awareness algorithm. In model-free undiscounted RL literature, there are mainly two types of methods for solving the training instability problem—policy search with no value function estimation [14], [15] or using prior task knowledge to tweak a discounted reward function so that the undiscounted objective is transformed to an alternative discounted return that expresses the same optimal policy. In this article, we aim to directly optimize the undiscounted return with RL and solve the instability problem without prior task knowledge. In the prior works, time-awareness is the only method that can directly optimize the undiscounted return [23]. Thus, the method was adopted as another baseline, and it was compared with LVS in terms of sampling efficiency. In the following content, the prefixes "LVS-" and "TA-" are,

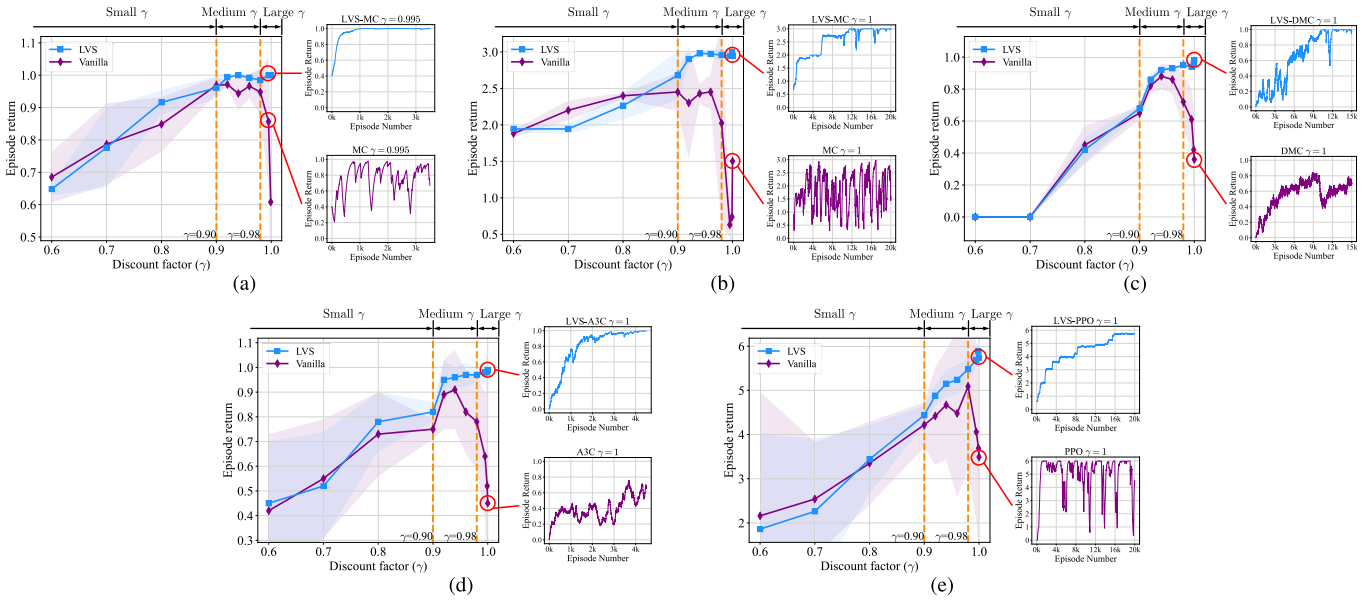

Fig. 8. Performance of the final policy under different choices of γ. The chosen γ are unevenly spaced, and the curves are divided into three regions, including 0.6, 0.7, 0.8, 0.9, 0.92, 0.94, 0.96, 0.98, and 0.995, 0.999, 1.0. For illustrating the training instability, we plot the learning curve of vanilla MC and LVS-MC under a large γ (0.995 for Cliff-Walking and 1 for other tasks). The learning curves are smoothed with a 0.99-exponential moving average. (a) Cliff-Walking. (b) Gathering. (c) ViZDoom navigation. (d) Jaco grasping. (e) YouBot covering.

respectively, used to denote the RL algorithms with LVS and time-awareness. Concretely, in the grid-world tasks, the state spaces are low dimensional; thus, adding 1-D time-step information to the state space is easy for the agent to notice the time information. Accordingly, we directly enlarge the state space as (s, t), where t is the step number, which is discrete. In the video game and the robot control tasks, the state spaces are high dimensional, for which the augmented time information seems to be "small." We adopt a classic method for the agent to "notice" the time information, which is to augment the time information by adding a channel that represents the time-step to the original RGB channels. This method can enhance the attention of the agent to the low-dimensional information in high-dimensional state spaces in many previous works.

3) Hyperparameter Settings: In the Cliff-Walking task and the Gathering task, we use Monte Carlo estimation with ϵ-greedy exploration method for LVS and baselines: 1) as for the baselines, the agent makes decision according to $a = \text{argmax}_a[Q(s, a)]$ with the probability $P(s, a_{\text{greedy}}) = \epsilon$ and a random action with the probability $P(s, a_{\text{random}}) = 1 - \epsilon$ and 2) as for LVS, the agent makes a decision according to $a = \text{argmax}_a[Q_{\text{rv}}(s, a) + Q_{\text{lv}}(s, a)]$ with the probability $P(s, a_{\text{greedy}}) = \epsilon$ and a random action with the probability $P(s, a_{\text{random}}) = 1 - \epsilon$. The Q-functions of the baselines and LVS are updated when every episode terminates. In the ViZDoom navigation task, we use Monte Carlo estimation with deep neural networks and the ε-greedy exploration method for LVS and baselines in this vision-based task: the sampling methods of the baselines and LVS are the same as those in the Cliff-Walking example. The Q-functions of the baseline and ours are updated when every episode terminates. In the Jaco grasping task and the Youbot covering task, we use actor–critic framework. As mentioned above, we use Monte Carlo estimation as the critic. The actions are sampled from the distribution of actions $\pi(a|s)$, which is represented by deep neural networks. These actors and critics are updated when every episode terminates. The main hyperparameters in our experiments include the learning rate and the exploration rate, which are determined from a grid search. The ranges of the hyperparameters for attempts are selected according to the popular choices in RL community [23], [31], [41], [42]. Specifically, the range of exploration rate in value-based RL algorithms [MC and deep Monte Carlo (DMC)] is [0.1, 0.6], and the range of coefficient of policy entropy in policy-based RL algorithms (A3C and PPO) is [0.0005, 0.02]. The range of learning rate in tabular algorithms (MC) is [0.005, 0.1], and the range of learning rate in algorithms with neural networks as function approximators (DMC, A3C, and PPO) is [0.00001, 0.01]. We discretized these ranges, tested the mean episode return of the LVS method over 100 episodes after training under each setting, and selected the one with the best performance. Tables I and II present the tried hyperparameters and the performances of the Gathering task and the Jaco Grasping task. Experiments show that our algorithm is robust to the selection of hyperparameters, and it can obtain good results for hyperparameters in a certain range. In the dreamer framework, we set the same learning rate of dynamics prediction as that of policy learning. The other hyperparameters are the same as those of [50], i.e., the imagination horizon is ten and the batch size is 50. In these five tasks, we set the same hyperparameters for the algorithms of LVS and the baselines. The learning rate and the exploration rate are shown in Table III.

C. Policy Performances Under Different Choices of γ

In this section, the training stability under different values of γ in Monte Carlo estimation is studied. With each choice of γ, ten training were performed under different random

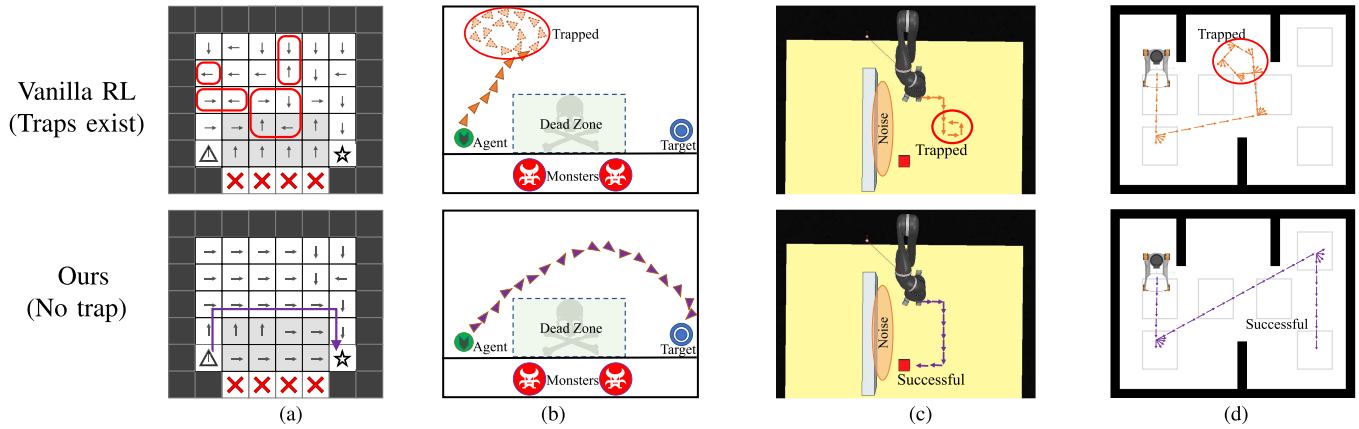

Fig. 9. Visualization of transient traps, when vanilla MC is used, and trajectories of LVS that do not suffer from transient traps. (a) Cliff-Walking. (b) ViZDoom navigation. (c) Jaco grasping. (d) Youbot covering.

TABLE I
HYPERPARAMETER TUNING (FOR GATHERING)

LVS Performance		Exploration Rate					
		0.1	0.2	**0.3**	0.4	0.5	0.6
Learning Rate	0.005	2.59	2.73	2.84	2.85	2.78	2.65
	0.007	2.66	2.88	2.94	2.91	2.82	2.71
	0.02	2.71	2.93	2.95	2.92	2.86	2.75
	0.05	2.75	2.95	**2.97**	2.94	2.89	2.82
	0.07	2.62	2.96	2.96	2.95	2.84	2.69
	0.1	2.66	2.88	2.89	2.79	2.70	2.46

TABLE II
HYPERPARAMETER TUNING (FOR JACO GRASPING)

LVS Performance		Exploration Rate (Policy Entropy Coefficient)					
		0.0008	0.002	**0.005**	0.008	0.01	0.02
Learning Rate	0.00001	0.87	0.89	0.94	0.92	0.9	0.88
	0.00005	0.91	0.97	0.95	0.96	0.93	0.89
	0.0001	0.88	0.96	**0.98**	0.98	0.96	0.91
	0.001	0.84	0.95	0.97	0.98	0.94	0.86
	0.005	0.82	0.87	0.93	0.95	0.92	0.83
	0.01	0.84	0.85	0.89	0.85	0.88	0.81

TABLE III
HYPERPARAMETER SETTINGS

Environments	Algorithms	Learning rate	Exploration rate
Cliff-Walking	MC	0.05	0.3
Gathering	MC	0.05	0.3
ViZDoom Navigation	DMC	0.0001	0.3
Jaco Grasping	A3C	0.0001	0.005(coefficient)
Youbot Navigation	PPO	0.0001	0.005(coefficient)

seeds, and the final performance was plotted as a function of γ in Fig. 8. The final performance was evaluated by averaging the episode return in the last 100 episodes in the ten tasks. The shaded area in the figure indicates the standard deviation. Although the agent is trained with $\gamma < 1$, it was evaluated with an undiscounted total reward. The behaviors of the agent can be divided into three regions, i.e., $\gamma < 0.9$, $0.9 < \gamma < 0.98$, and $\gamma > 0.98$. For $\gamma < 0.9$, the discount rate was too small to satisfy the optimality, and the performance increased with γ; for $0.9 < \gamma < 0.98$, the performance did not continue increasing with γ due to the training instability; for $\gamma > 0.98$, the learning process became seriously unstable, and the performance degraded fast. To concretely illustrate the training oscillation, the learning curves of $\gamma = 0.995$ and $\gamma = 1.0$ were plotted. By contrast, the performance of the LVS policy increased stably even if $\gamma > 0.9$, indicating the effectiveness of our method.

D. Visualization of Transient Traps

In this section, the transient trap problem that occurs in the learning process under a large γ is illustrated. Without loss of generality, γ was set to 1. In the Cliff-Walking task, Crude Monte Carlo [20] was taken as the base algorithm, and the agent's policy was visualized at a late period in the training process. The policy is marked as arrows in Fig. 9(a), and the agent's greedy action is shown in each state. As marked with orange circles, there are many traps in the environment. If the agent moves into one of these traps, the agent cannot move out and receive zero rewards. In contrast, as shown in Fig. 9(a), there were no traps when the LVS-MC method was used, and the agent can successfully reach the target state.

Next, the ViZDoom navigation, the Jaco grasping, and the Youbot covering tasks were used to show the trap problem in the tasks with continuous state spaces. For these tasks, DMC, REINFORCE-AC, and REINFORCE-PPO algorithms were, respectively, taken as the base algorithm, and the agent's trajectories were visualized in a late period of the training process. The trajectories are illustrated as a series of arrows in Fig. 9(b)–(d). As illustrated by the annotations, the trajectories contain cycles in which the agent fails to move out and complete the task. In contrast, as shown in Fig. 9(b)–(d), the trajectories do not contain cycles, and the agent can complete the tasks successfully.

The above results indicate the existence of transient traps in the environment with discrete state space and the environment with continuous state space. Also, the effectiveness of the LVS method in eliminating the transient traps is demonstrated.

E. Effectiveness in Improving Training Stability

The training curves and numeric results are presented, and meanwhile, the overall performance of the LVS method is

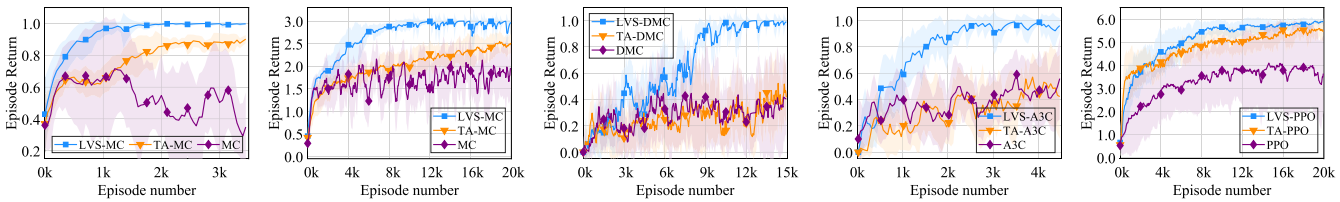

Fig. 10. Learning curves of the model-free algorithms. (a) Cliff-Walking. (b) Gathering. (c) ViZDoom navigation. (d) Jaco grasping. (e) YouBot covering.

TABLE IV
TRAINING RESULTS (MEAN ± STANDARD DEVIATION)

Type	Algorithms	Cliff-Walking	Gathering	ViZDoom navigation	Jaco grasping	YouBot covering
Model-free	Vanilla	0.35±0.46	1.82±1.07	0.36±0.48	0.51±0.50	3.67±1.81
	TA	0.89±0.10	2.48±0.34	0.45±0.50	0.42±0.49	5.54±0.59
	LVS	**1.00±0.01**	**2.97±0.29**	**0.98±0.13**	**0.98±0.15**	**5.86±0.23**
Model-based	Dreamer	0.69±0.46	2.01±0.90	0.65±0.48	0.73±0.45	4.11±2.02
	TA	0.99±0.09	2.59±0.81	0.83±0.38	0.81±0.39	5.00±0.00
	LVS	**1.00±0.00**	**3.00±0.03**	**1.00±0.05**	**0.99±0.08**	**6.00±0.00**

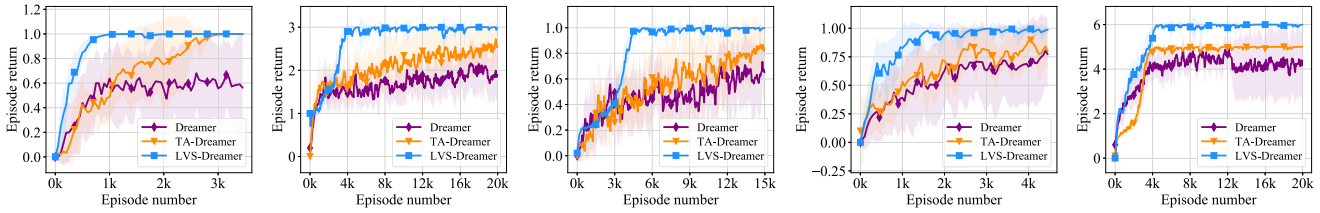

Fig. 11. Learning curves of the model-based algorithms. (a) Cliff-Walking. (b) Gathering. (c) ViZDoom navigation. (d) Jaco grasping. (e) YouBot covering.

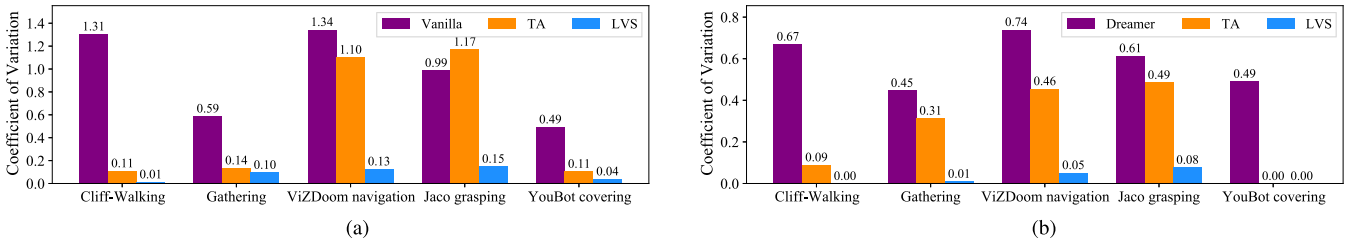

Fig. 12. CV of the basic algorithm, TA, and LVS in all the five tasks. (a) Model-free algorithms. (b) Model-based algorithms.

compared with that of the baselines. The training curves of the five tasks are plotted in Figs. 10(a)–(e) and 11(a)–(e). Each curve and its shaded region represent the mean episode reward and standard deviation of the mean, respectively, averaged over ten experiments with different random seeds. The curves are drawn by selecting the data at equal intervals of 50 and are smoothed with a 0.7-exponential moving average for clarity. The final performance was calculated by averaging the episode return of the last 100 episodes, and the results are shown in the format of mean ± standard deviation (see Table IV). It can be seen that, in all the tasks, the training of base algorithms presents a large variance, while the training of algorithms with the TA and LVS method is more stable. Also, the algorithms with the TA method completed training slowly, and the final performances were worse than that of the algorithms with the LVS method.

To illustrate the convergence performance, we use the coefficient of variation (CV) [51] as a normalized measure of the dispersion of probability distribution, which is defined as the ratio of standard deviation to mean. As shown in Fig. 12, the CVs of the LVS algorithm are smaller than 0.2 in all the five tasks, which demonstrates strong stability of training. Furthermore, the CVs of the LVS algorithm are lower than those of the compared methods, especially in complex tasks, including the ViZDoom navigation and the Jaco grasping. This shows that LVS achieves better training stability than the compared methods. Overall, the algorithms with LVS achieve the best training stability and final performance among the comparing methods. It should be noted that the above results are obtained from different types of tasks. The Gathering and Youbot covering are tasks with multiple rewards, while the others are tasks with a single positive reward. Especially, the

ViZDoom, Jaco, and Youbot tasks have continuous state space, while the others have discrete state space. The algorithms used in Jaco and Youbot tasks employ stochastic policies, while the algorithms used in the other three tasks employ deterministic policies. These results indicate that the effectiveness of the LVS method in stabilizing training is independent of the considered form of the reward function, state space, and policy.

VII. Conclusion and Discussion

The transient trap problem can cause training instability in undiscounted RL. This article analyzes the transient trap problem and proposes the LVS method to eliminate transient traps and improve the training stability. Also, the analysis of the existence condition of transient trap in this article can explain the training instability in discounted RL with large discount factors: the bigger the γ, the weaker its ability to decay the values of inconsistently selected actions. Meanwhile, it can cause the value of nonoptimal actions to be close to that of the optimal actions, thus forming transient traps when there are noises in the training process. For this problem, the LVS method should also be conducive to improve training stability.

In future work, the convergence of our method will be analyzed and verified theoretically. One difficulty is that the undiscounted RL is not a contraction mapping. However, the experimental results in this article have shown that our method has good convergence and can achieve ideal results in various tasks. Besides, we plan to relax the assumption of a deterministic policy in the theoretical analysis in future work because the effectiveness of the LVS method in stochastic algorithms has been shown empirically.

References

[1] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017.

[2] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[3] J. Ren, S. Guo, and F. Chen, "Orientation-preserving Rewards' balancing in reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 11, 2021, doi: 10.1109/TNNLS.2021.3080521.

[4] S. Pateria, B. Subagdja, A.-H. Tan, and C. Quek, "End-to-end hierarchical reinforcement learning with integrated subgoal discovery," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 22, 2021, doi: 10.1109/TNNLS.2021.3087733.

[5] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 1–8.

[6] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," in *Proc. Robot., Sci. Syst. XIII*, Jul. 2017, pp. 1–10.

[7] D. Liu, H. Li, and D. Wang, "Error bounds of adaptive dynamic programming algorithms for solving undiscounted optimal control problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 6, pp. 1323–1334, Jun. 2015.

[8] W. Zhao, H. Liu, and F. L. Lewis, "Robust formation control for cooperative underactuated quadrotors via reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 10, pp. 4577–4587, Oct. 2020.

[9] G. Peng, C. L. P. Chen, and C. Yang, "Neural networks enhanced optimal admittance control of robot-environment interaction using reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 2, 2021, doi: 10.1109/TNNLS.2021.3057958.

[10] Y. Hu, W. Wang, H. Liu, and L. Liu, "Reinforcement learning tracking control for robotic manipulator with kernel-based dynamic model," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3570–3578, Sep. 2020.

[11] P. Jiang, S. Song, and G. Huang, "Attention-based meta-reinforcement learning for tracking control of AUV with time-varying dynamics," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, May 25, 2021, doi: 10.1109/TNNLS.2021.3079148.

[12] A. Camacho, C. J. Muise, and S. A. McIlraith, "From FOND to robust probabilistic planning: Computing compact policies that bypass avoidable deadends," in *Proc. 26th Int. Conf. Automated Planning Scheduling*, 2016, pp. 65–69.

[13] Z. Xu, H. P. van Hasselt, M. Hessel, J. Oh, S. Singh, and D. Silver, "Meta-gradient reinforcement learning with an objective discovered online," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2020, pp. 15254–15264.

[14] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017, *arXiv:1703.03864*.

[15] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," 2018, *arXiv:1803.07055*.

[16] Z. Cao, H. Guo, J. Zhang, F. A. Oliehoek, and U. Fastenrath, "Maximizing the probability of arriving on time: A practical Q-learning method," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 4481–4487.

[17] C. Tessler and S. Mannor, "Reward tweaking: Maximizing the total reward while planning for short horizons," 2020, *arXiv:2002.03327v2*.

[18] Z. Xu, H. van Hasselt, and D. Silver, "Meta-gradient reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2402–2413.

[19] K. D. Asis, A. Chan, S. Pitis, R. S. Sutton, and D. Graves, "Fixed-horizon temporal difference methods for stable reinforcement learning," in *Proc. 34th AAAI Conf. Artif. Intell.*, 2020, pp. 3741–3748.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[21] A. Hashavit and S. Markovitch, "Max-prob: An unbiased rational decision making procedure for multiple-adversary environments," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 222–227.

[22] A. Kolobov, Mausam, D. S. Weld, and H. Geffner, "Heuristic search for generalized stochastic shortest path MDPs," in *Proc. 21st Int. Conf. Automated Planning Scheduling*, 2011, pp. 1–8.

[23] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev, "Time limits in reinforcement learning," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 4042–4051.

[24] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[25] H. van Seijen, M. Fatemi, and A. Tavakoli, "Using a logarithmic mapping to enable lower discount factors in reinforcement learning," in *Annu. Conf. Neural Inf. Process. Syst.*, 2019, pp. 14111–14121.

[26] C. Dann and E. Brunskill, "Sample complexity of episodic fixed-horizon reinforcement learning," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 2818–2826.

[27] R. Amit, R. Meir, and K. Ciosek, "Discount factor as a regularizer in reinforcement learning," in *Proc. 37th Int. Conf. Mach. Learn.*, vol. 119, 2020, pp. 269–278.

[28] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "ViZDoom: A doom-based AI research platform for visual reinforcement learning," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Sep. 2016, pp. 1–8.

[29] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 1321–1326.

[30] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2140–2146.

[31] H. Gao, Z. Yang, X. Su, T. Tan, and F. Chen, "Adaptability preserving domain decomposition for stabilizing Sim2Real reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 4403–4410.

[32] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, "Generating adjacency-constrained subgoals in hierarchical reinforcement learning," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2020, pp. 21579–21590.

[33] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Denver, CO, USA, 1995, pp. 1–7.

[34] A. Schwartz, "A reinforcement learning method for maximizing undiscounted rewards," in *Proc. 10th Int. Conf. Mach. Learn.*, 1993, pp. 298–305.

[35] D. P. Bertsekas and J. N. Tsitsiklis, "An analysis of stochastic shortest path problems," *Math. Oper. Res.*, vol. 16, no. 3, pp. 580–595, Aug. 1991.

[36] H. Yu and D. P. Bertsekas, "On boundedness of Q-learning iterates for stochastic shortest path problems," *Math. Oper. Res.*, vol. 38, no. 2, pp. 209–227, May 2013.
[37] N. Jiang, A. Kulesza, S. P. Singh, and R. L. Lewis, "The dependence of effective planning horizon on model accuracy," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 4180–4189.
[38] A. Kolobov, Mausam, and D. S. Weld, "A theory of goal-oriented MDPs with dead ends," in *Proc. 28th Conf. Uncertainty Artif. Intell.*, 2012, pp. 438–447.
[39] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Mach. Learn.*, vol. 22, nos. 1–3, pp. 123–158, Mar. 1996.
[40] M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas, and R. Munos, "Increasing the action gap: New operators for reinforcement learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, D. Schuurmans and M. P. Wellman, Eds., 2016, pp. 1476–1483.
[41] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a GPU," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–12.
[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[43] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 2778–2787.
[44] S. S. Ruan, G. Comanici, P. Panangaden, and D. Precup, "Representation discovery for MDPs using bisimulation metrics," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 3578–3584.
[45] N. Ferns, P. Panangaden, and D. Precup, "Metrics for finite Markov decision processes," in *Proc. 20th Conf. Uncertainty Artif. Intell.*, 2004, pp. 162–169.
[46] P. Dayan, "Improving generalization for temporal difference learning: The successor representation," *Neural Comput.*, vol. 5, no. 4, pp. 613–624, Jul. 1993.
[47] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 2371–2378.
[48] Y. Zhu *et al.*, "Reinforcement and imitation learning for diverse visuomotor skills," in *Proc. Robot., Sci. Syst. XIV*, Jun. 2018, pp. 1–12.
[49] C. Gordón, P. Encalada, H. Lema, D. León, C. Castro, and D. Chicaiza, "Intelligent autonomous navigation of robot KUKA YouBot," in *Proc. Intell. Syst. Conf.*, vol. 1038, 2019, pp. 954–967.
[50] D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *Int. Conf. Learn. Represent.*, 2020, pp. 1–20.
[51] Z. Jalilibal, A. Amiri, P. Castagliola, and M. B. C. Khoo, "Monitoring the coefficient of variation: A literature review," *Comput. Ind. Eng.*, vol. 161, Nov. 2021, Art. no. 107600.

Haichuan Gao received the B.Eng. degree from the Department of Mechanics, Central South University, Changsha, China, in 2018. He is currently pursuing the Ph.D. degree in control science and engineering with Tsinghua University, Beijing, China.

His current research interests include reinforcement learning and robot learning.

Zhile Yang received the B.Eng. and M.S. degrees from the Department of Automation, Tsinghua University, Beijing, China, in 2018 and 2021, respectively.

His current research interests include reinforcement learning and computational neuroscience.

Tian Tan received the B.S. degree in telecommunications engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2012, and the M.S. and Ph.D. degrees in engineering and the Ph.D. degree minor in computer science from Stanford University, Stanford, CA, USA, in 2015 and 2020, respectively.

His research interests broadly include topics in machine learning and algorithms, such as reinforcement learning, multitask learning, gradient boosting decision trees, deep learning, and statistical learning theory.

Tianren Zhang received the B.Eng. degree from the Department of Automation, Tsinghua University, Beijing, China, in 2019, where he is currently pursuing the M.S. degree.

His current research interests include reinforcement learning and general machine learning.

Jinsheng Ren received the B.S. degree in automation from the University of Electronic Science and Technology of China, Chengdu, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Automation, Tsinghua University, Beijing, China.

His current research interests include computer vision, artificial intelligence, lifelong learning, reinforcement learning, and learning theory.

Pengfei Sun received the B.Eng. degree in automation from Northeastern University, Shenyang, China, in 2016. He is currently pursuing the Ph.D. degree with the Department of Automation, Tsinghua University, Beijing, China.

His current research interests include computer vision, artificial intelligence, robot learning, and learning theory.

Shangqi Guo received the B.S. degree in mathematics and physics basic science from the University of Electronic Science and Technology of China, Chengdu, China, in 2015, and the Ph.D. degree from the Department of Automation, Tsinghua University, Beijing, China, in 2021.

His current research interests include inference in artificial intelligence, brain-inspired computing, and reinforcement learning.

Feng Chen (Member, IEEE) received the B.S. and M.S. degrees in automation from Saint-Petersburg Polytechnic University, Saint Petersburg, Russia, in 1994 and 1996, respectively, and the Ph.D. degree from the Department of Automation, Tsinghua University, Beijing, China, in 2000.

He is currently a Professor with Tsinghua University. His current research interests include computer vision, brain-inspired computing, and inference in graphical models.