

Segmenting Action-Value Functions Over Time-Scales in SARSA via TD(Δ)

Mahammad Humayoo^{1,2,3,4,*}

¹Hanshan Normal University, Chaozhou, Guangdong, China 521041

²CAS Key Laboratory of Network Data Science and Technology, Institute of Computing Technology, CAS, Beijing, China 100190

³University of Chinese Academy of Sciences, Beijing, China 101408

⁴School of Computer Science, Beijing Institute of Technology, Beijing, China 100081

ABSTRACT

In numerous episodic reinforcement learning (RL) environments, SARSA-based methodologies are employed to enhance policies aimed at maximizing returns over long horizons. Traditional SARSA algorithms face challenges in achieving an optimal balance between bias and variation, primarily due to their dependence on a single, constant discount factor (η). This investigation enhances the temporal difference decomposition method, TD(Δ), by applying it to the SARSA algorithm, now designated as SARSA(Δ). SARSA is a widely used on-policy RL method that enhances action-value functions via temporal difference updates. By splitting the action-value function down into components that are linked to specific discount factors, SARSA(Δ) makes learning easier across a range of time scales. This analysis makes learning more effective and ensures consistency, particularly in situations where long-horizon improvement is needed. The results of this research show that the suggested strategy works to lower bias in SARSA's updates and speed up convergence in both deterministic and stochastic settings, even in dense reward Atari environments. Experimental results from a variety of benchmark settings show that the proposed SARSA(Δ) outperforms existing TD learning techniques in both tabular and deep RL environments.

1 Introduction

Reinforcement learning (RL) agents encounter significant challenges in the optimization of long-horizon rewards. Traditional methods employing singular discount variables encounter inherent challenges: small η results in low-variance but myopic policies, whereas large η suffers from excessive variance. Fixed discounts remain constant despite variations in reward densities¹⁻³. Temporal difference (TD) learning methods, such as Q-learning and SARSA, have shown effectiveness in various tasks by enabling agents to estimate action-value functions that predict expected future rewards. Traditionally, these strategies use a discount factor of $0 \leq \eta < 1$, with values getting closer to $\eta = 0$ to put more weight on short-term rewards than long-term ones. This approach shortens the planning horizon and makes learning more stable and efficient². Prokhorov and Wunsch⁴ show that discount factors $\eta < 1$ often lead to better results in the early stages of learning. In certain scenarios, such as long-horizon tasks^{5,6} where long-term planning is crucial—like navigation or decision-making tasks that involve delayed rewards—this approach can introduce bias, complicating the agent's ability to determine the most effective long-horizon policies.

For example, SARSA² (State-Action-Reward-State-Action) is an on-policy RL algorithm that updates the action-value function $Q(s,a)$ according to the state-action transitions the agent sees. The standard SARSA algorithm works well in environments with shorter time horizons; however, it faces challenges when utilized for tasks with longer horizons. The main reason for the observed phenomenon is the use of a single discount factor η , which requires a trade-off between bias and variance. This often leads to instability and inefficiency during the learning process. A variety of recommendations exist for addressing this issue^{1,4,6-10}.

Through the use of weighted combinations of value functions, Fedus et al.¹⁰ presented the concept of hyperbolic discounting. However, their formulation is restricted to state-value functions and does not extend to action-value functions. Meta-gradient RL⁸ is a technique that enables agents to adapt their learning processes dynamically. Meta-gradient means finding or improving targets and hyperparameters during training, which could make learning more efficient and adaptable. A gradient-based meta-learning algorithm capable of online adaptation to the true value function while engaging with and learning from the environment. However, akin to other methodologies, meta-gradient RL primarily emphasizes state-value functions, rather than action-value formulations.

The work that is most relevant in this case is by Romoff et al.¹, which provides a theoretical basis for temporal difference decomposition using the TD(Δ) framework. TD(Δ) segments the state-value function $V(s)$ into components using different discount factors, also known as TD(Δ). This lets the agent learn about short-term and long-term returns separately, which can

then be combined to improve performance. This decomposition framework demonstrates considerable potential in addressing the bias-variance trade-off that is intrinsic to temporal difference learning. However, $TD(\Delta)$ is constrained to state-value functions, which presents a limitation in the application of multi-timescale representations to action-value-based algorithms such as SARSA and Q-learning.

This article proposes SARSA(Δ), an enhancement of the $TD(\Delta)$ approach integrated into the SARSA algorithm, based on this principle. $TD(\Delta)$ emphasizes the actor-critic framework and conventional TD learning, whereas SARSA(Δ) divides the state-action value function into multiple partial estimators corresponding to various discount factors, known as delta estimators. These estimators give a rough estimate of the difference $D_m(s, a) = Q_{\eta_m}(s, a) - Q_{\eta_{m-1}}(s, a)$ between action-value functions. This makes learning faster in contexts where actions yield long-term effects. Like $TD(\Delta)$, SARSA(Δ) aims to enhance a collection of delta estimators, with each estimator linked to a specific discount factor. Smaller discount factors lead to quicker convergence via segmentation, whereas larger discount factors enhance this foundation, facilitating improved long-term planning.

This paper provides a multi-stage SARSA(Δ) updating algorithm that separates action-value functions, which allows for faster convergence and improved policy efficacy, especially in complex deterministic and stochastic environments like Atari games. Moreover, our research shows that SARSA (Δ) can seamlessly accommodate various multi-step RL methodologies, such as n-step SARSA and eligibility traces. This study emphasizes the benefits of SARSA(Δ) in various tasks, including a basic ring MDP used by Kearns & Singh¹¹. It outperforms conventional SARSA in scenarios requiring a balance between short-term action consequences and long-term reward maximization. SARSA (Δ) may adjust discount factors for multiple time scales, making it valuable where time scale is precious.

The reason for extending $TD(\Delta)$ to SARSA(Δ) is that temporal difference ($TD(\Delta)$) learning provides a way to break down value functions across a range of time scales, which solves the problems that come up when learning on a single scale for long-horizon scenarios. We want to break down the action-value function $Q(s,a)$ into components that focus on different discount factors by extending $TD(\Delta)$ to SARSA(Δ). This decomposition enables SARSA to leverage the advantages of learning over multiple time scales, which stabilizes the learning process and reduces the bias-variance trade-off. This add-on gives you a more detailed and scalable way to learn the action-value function in complex environments. Table 1 presents a comparison of the key features of $TD(\Delta)$ /SARSA(Δ) and Standard TD /SARSA-Learning.

Table 1. Comparison of the essential properties of $TD(\Delta)$ and SARSA(Δ).

Feature	$TD(\Delta)$ ¹	SARSA(Δ)-Learning (proposed by us)	Standard TD ¹²	SARSA-Learning ¹²
Value function decomposition	Yes (by η)	No	No	No
Action-value function decomposition	No	Yes (by η)	No	No
Handles multiple time-scales	Yes	Yes	No	No
Learning stability	Enhanced, particularly long-term	Enhanced, particularly long-term	Reduced for long-term rewards	Reduced for long-term rewards
Scalability	Improved	Improved	Restricted by variance/bias	Restricted by variance/bias

Recent research on value decomposition¹ has demonstrated potential for state-value functions; however, it identifies three significant shortcomings that this work aims to address: (i) Action-Value Formulation: No current method effectively decomposes $Q(s, a)$ while maintaining on-policy guarantees. This study extends $TD(\Delta)$ to SARSA(Δ) by decomposing the action-value function $Q(s,a)$ into delta components across multiple time scales. (ii) Theoretical Foundations: Current analyses do not address the bias-variance trade-off concerning action values. (iii) The proposed SARSA(Δ) algorithm improves learning efficiency and stability through the independent learning of components corresponding to different discount factors, which are subsequently integrated to form the comprehensive action-value function. (iv) Both theoretical and empirical evaluations have shown that this multi-scale breakdown has a number of benefits, particularly in environments where there are long-horizon rewards, as demonstrated in Table 1.

The subsequent portions of the paper are organized as follows: Section 2 presents a discussion on relevant literature. Section 3 provides the essential background information. The principles of SARSA(Δ) and theoretical analysis are shown in Sections 4 and 5, respectively. Section 6 provides a detailed summary of the experiments performed. Ultimately, we present a conclusion in section 7.

2 Related Work

Optimizing for long-horizon rewards in reinforcement learning (RL) is difficult owing to the complexities of learning with undiscounted returns. Temporal discounting is often used to simplify this process; however, it can introduce bias.

2.1 Temporal Decomposition Approaches

In their investigation, Romoff et al.¹ explored the differentiation of value functions across multiple time scales by scrutinizing individuals with diminished discount factors. The system became more efficient and simpler to expand. In RL, Sherstan et al.¹³

introduced γ -nets, a method designed to enhance value function estimates across various timescales. Authors' recent work^{10,14} showed how to add hyperbolic discounting to RL. Exponential discounting was used in traditional RL, but this didn't match the hyperbolic discounting seen in the behavior of human and animals. The authors showed an agent that used temporal-difference learning methods to get close to hyperbolic discount functions. The research also found that learning value functions across multiple time horizons increases performance, particularly when used as an auxiliary task in value-based RL algorithms like Rainbow. This approach has shown potential in risky and uncertain environments.

Recent research^{4,6-8,15-17} focuses on the exact selection of the discount factor. Xu¹²⁴ et al.⁸ proposed meta-gradient methods for choosing discount factors in RL that change over time. They stressed the importance of balancing short- and long-term rewards, which is similar to why we break down action-value functions across multiple discount factors in SARSA(Δ). Lastly, hierarchical RL and the bias-variance trade-off are part of a large body of research that is related to our study in an indirect way. These studies¹⁸⁻²⁴ introduced the idea of hierarchical RL breaking value functions down into smaller tasks, similar to how TD(Δ) broke value functions down across different time scales. These results backed up the claim that breaking down value functions can make learning more effective and stable.

2.2 Multi-Scale RL

To address issues in RL, especially when the optimum value function is complex and poorly represented by conventional DRL methods, van Seijen et al.²⁴ introduced Hybrid Reward Architectures (HRA). The HRA was designed to enhance learning consistency and effectiveness by partitioning the reward function and allocating distinct value functions to each component. Efficient multi-horizon learning in off-policy RL aims to improve agents' ability to make predictions and decisions over a wide range of time scales. This is important for navigating through complex and unpredictable environments. Ali¹⁴¹ et al.^{25,26} investigated designs that enable agents to learn value estimates over several horizons at the same time. This improves their capacity to adjust and flourish in novel environments. In the realm of artificial intelligence and robotics, coordinating agents with limited communication and across diverse planning periods is a significant challenge to multi-horizon, multi-agent planning.

Seiler et al.²⁷ introduced Decentralized Monte Carlo Tree Search (Dec-MCTS) and its variants, which have arisen as formidable instruments for addressing these challenges, allowing agents to plan effectively and adaptively in dynamic, unpredictable environments. Benechhab et al.²⁸ came up with a novel technique in model-based RL (MBRL) called multi-timestep models, aimed at mitigating the problem of escalating prediction errors over extended simulated trajectories. By optimizing for several future steps instead of only one-step-ahead forecasts, these models enhance the precision and resilience of long-term predictions, especially in complex or chaotic environments.

Bonnet et al.²⁹ presented multi-step meta-gradient RL, which uses meta-gradients that have been built up across many stages instead of just one. This makes learning algorithms more flexible. This method may provide richer and more useful learning signals, but it does come with some significant trade-offs. But it also increases computational complexity and variation, which may hinder performance if not handled carefully. While deep RL (DRL) can teach agents difficult tasks, it still struggles to improve sampling efficiency and adapt to changing environments. Eligibility traces, a well-established RL approach, expedite learning, but parameter dependencies make their integration with deep neural networks difficult. Kobayashi³⁰ introduced several time-scale eligibility traces to equilibrate short-term and long-term credit assignment. This technique improves learning speed and policy quality in online DRL environments by substituting the most important adaptively gathered traces.

2.3 SARSA Variants and Traditional RL Methods

Expected SARSA⁹ is a kind of SARSA that speeds up learning by exploiting information from a stochastic behavior policy to create updates that are less variance. Many RL algorithms, like SARSA, employ temporal-difference (TD) learning, which looks at the differences between two predictions made one after the other instead of the final result. Sutton³¹ came up with this notion for the first time in 1988. The research shows that TD approaches may be useful for predicting jobs, especially where incremental learning is crucial, and that they can even operate jointly in certain cases.

Theoretical convergence for function approximation-based TD learning algorithms was shown by Tsitsiklis and Van Roy³². If you follow the SARSA update rule for delta component updates, this research may be used with SARSA(Δ) to get the same convergence properties. None of the previously mentioned research discussed short-term estimations to train long-term action-value functions. One benefit is that you may ask about shorter time periods and utilize our method as a generic action-value function. Separating action-value functions across time scales using TD(Δ) in SARSA makes a big difference in performance. Theoretical work by Kearns and Singh¹¹ helped us comprehend the bias-variance trade-off that comes up when we break down Q-values over multiple time scales. It also set limits on the bias and variance of TD updates.

Most of the methods listed above focus on state values, although action-value decomposition has not yet been looked at. Our proposed strategy focuses on breaking down $Q(s, a)$ over different time scales, which makes long-term learning more stable.

3 Background and notation

Examine a completely observable Markov Decision Process (MDP)³³, characterized by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, where \mathcal{S} denotes the state space, \mathcal{A} signifies the action space, and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \rightarrow [0, 1]$ represents the transition probabilities that associate state-action pairings with distributions across subsequent states, whereas $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ denotes the reward function. At each timestep n , the agent is state s_n , selects an action a_n , receives a reward $r_n = r(s_n, a_n)$, and transitions to the next state $s_{n+1} \sim \mathcal{P}(\cdot | s_n, a_n)$.

Within a standard MDP framework, an agent tries to get the highest possible discounted return, which is defined as $Q_\eta^\pi(s, a) = \mathbb{E}[\sum_{n=0}^{\infty} \eta^n r_{n+1} | s_n = s, a_n = a]$, where η is the discount factor and $\pi : \mathcal{S} \rightarrow \mathcal{A} \rightarrow [0, 1]$ stands for the policy that the agent follows. The action-value function $Q_\eta^\pi(s, a)$ is determined as the fixed point of the Bellman operator $\mathcal{T}Q = r^\pi + \eta \mathcal{P}^\pi Q$, where r^π denotes the expected immediate reward and \mathcal{P}^π represents the transition probability operator associated with the policy π . For convenience, we remove the superscript π for the rest of the paper.

Using temporal difference (TD)³⁴ learning, the action-value estimate \hat{Q}_η can approximate the true action-value function Q_η . The one-step TD error $\delta_n^\eta = r_{n+1} + \eta \hat{Q}_\eta(s_{n+1}, a_{n+1}) - \hat{Q}_\eta(s_n, a_n)$ is used to update the action-value function given a transition (s_n, a_n, r_n, s_{n+1}) .

An on-policy RL technique called SARSA learns the action-value function $Q(s, a)$ for a given policy π . The action-value function represents the expected total reward derived from state s , executing action a , and adhering to policy π . Then the SARSA update rule is expressed as follows:

$$Q(s_n, a_n) \leftarrow Q(s_n, a_n) + \alpha [r_n + \eta Q(s_{n+1}, a_{n+1}) - Q(s_n, a_n)] \quad (1)$$

Here, α is the learning rate, η is the discount factor, and r_n is the immediate reward. Long-horizon tasks present challenges for SARSA since the choice of η can compromise learning efficiency with appropriate long-term reward maximizing.

On the other hand, for a complete trajectory, we can employ the discounted sum of one-step TD errors, generally known as the λ -return³⁴ or, equivalently, the Generalized Advantage Estimator (GAE)³⁵. The GAE improves advantage estimates by balancing the trade-off between variance and bias through the parameters λ and η . The Generalized Advantage Estimator is typically represented by the following equation:

$$A(s_n, a_n) = \sum_{k=0}^{\infty} (\lambda \eta)^k \delta_{t+k}^\eta \quad (2)$$

Where δ_{n+k} is the TD error at time n , computed as follows: $\delta_{n+k} = r_{n+k} + \eta Q(s_{n+k+1}, a_{n+k+1}) - Q(s_{n+k}, a_{n+k})$.

Loss function for Q-Value estimation utilizing GAE. The loss function $\mathcal{L}(\theta)$ for approximating the Q-value function is defined as the mean squared error between the current Q-value estimate $Q(s_n, a_n; \theta)$ and the target Q-value adjusted by the advantage estimator. Thus, it is possible to concisely write the loss function using $A(s_n, a_n)$ and $Q(s_n, a_n)$ as follows:

$$\mathcal{L}(\theta) = \mathbb{E} \left[\left(Q(s_n, a_n; \theta) - \left(Q(s_n, a_n) + A(s_n, a_n) \right) \right)^2 \right] \quad (3)$$

Despite SARSA being on-policy and lacking an explicit policy update, policy selection in SARSA can still be influenced by Eq. 4. For actor-critic architectures^{36–38}, the action-value function is updated according to Eq. 3, and a stochastic parameterized policy (actor, $\pi_v(a|s)$) is learned from this value estimator through the advantage function, with the loss being:

$$\mathcal{L}(v) = \mathbb{E} \left[-\log \pi(a|s; v) A(s, a) \right] \quad (4)$$

Proximal Policy Optimization (PPO)³⁹ is an improvement on actor-critic methods. It limits policy updates to a specific area of optimization called a trust region. It does this by using a clipping objective that compares the current parameters, v , to the old parameters, v_{old} :

$$\mathcal{L}(v) = \mathbb{E} \left[\min \left(\rho(v) A(s, a), \psi(v) A(s, a) \right) \right] \quad (5)$$

In on-policy approaches such as SARSA, the probability ratio $\rho(v)$ between the current policy and a preceding policy is often used. SARSA evaluates actions in accordance with the current policy, concurrently updating the action-value function as actions are performed. However, when utilized within an actor-critic framework, the SARSA-learning agent may adopt a similar policy ratio:

$$\rho(v) = \frac{Q(s, a; v)}{Q(s, a; v_{old})} \quad (6)$$

where $\psi(v) = \text{clip}(\rho, 1 - \epsilon, 1 + \epsilon)$ represents the clipped likelihood ratio, and $\epsilon < 1$ is a negligible parameter employed to constrain the update. The loss function for the policy update can be articulated as follows:

$$\mathcal{L}(v) = \mathbb{E} \left[\min \left(\rho(v)A(s, a), \text{clip}(\rho(v), 1 - \epsilon, 1 + \epsilon)A(s, a) \right) \right] \quad (7)$$

4 Methodology

The TD(Δ) paradigm boosts regular TD learning by splitting action-value functions over several discount factors, $\eta_0, \eta_1, \dots, \eta_m$. This breakdown makes it easier to find the action-value function as a sum of delta estimators, where each estimator shows how action-value functions differ at different discount factors. The main advantage of this method is that it makes it easier to control learning variance and bias by focusing on smaller, more controllable time scales.

4.1 TD(Δ) Framework

To compute D_m for SARSA as outlined in the study, it is essential to comprehend the mechanism by which delta estimators (D_m) approximate the discrepancies between action-value functions across successive discount factors. This approach for SARSA will be outlined in a stepwise manner.

Delta Estimators (D_m) and Action-Value Functions ($Q(s, a)$): Delta estimators, denoted as D_m , quantify the variation between action-value functions associated with consecutive discount factors:

$$D_m = Q_{\eta_m} - Q_{\eta_{m-1}} \quad (8)$$

In this context, Q_{η_m} denotes the action-value function with discount factor η_m , but $Q_{\eta_{m-1}}$ signifies the action-value function with the prior discount factor, η_{m-1} .

4.2 Single-Step TD (SARSA(Δ))

This section initiates with an overview of the delta estimators (D_m) pertaining to the action-value functions Q employed in SARSA. SARSA is expanded to TD(Δ), which is called SARSA(Δ). SARSA(Δ) aims to determine the action-value function $Q(s, a)$, representing the expected return of adhering to policy π from state s , executing action a , and subsequently continuing to follow π . The goal is to optimize the total reward by improving the action-value function over various time scales. SARSA(Δ) fundamentally decomposes the action-value function $Q(s, a)$ into multiple delta components $D_m(s, a)$, each associated with a distinct discount factor η_m . The relationship between these delta components (i.e., delta function) is expressed in the following manner:

$$D_m(s, a) := Q_{\eta_m}(s, a) - Q_{\eta_{m-1}}(s, a) \quad (9)$$

where $\eta_0, \eta_1, \dots, \eta_m$ represent the discount factors across various time-scales, and define $D_0(s, a) := Q_{\eta_0}(s, a)$. The action-value function $Q_{\eta_m}(s, a)$ is defined as the cumulative sum of all D-components up to m:

$$Q_{\eta_m}(s, a) = \sum_{x=0}^m D_x(s, a) \quad (10)$$

The policy dictates how the state-action pair in conventional SARSA determines the update rule. The TD error adjusts the action-value function, and the policy determines the action in the following state:

$$Q(s_n, a_n) \leftarrow Q(s_n, a_n) + \alpha [r_n + \eta Q(s_{n+1}, a_{n+1}) - Q(s_n, a_n)] \quad (11)$$

In this context, r_n signifies the reward acquired from performing action a_n in state s_n , whereas (s_{n+1}, a_{n+1}) denotes the ensuing state-action pair. All of the delta components $D_n(s, a)$ are updated separately using the SARSA rule. The modifications to each component use the same structure as the standard SARSA method, but they are changed to incorporate the delta function at each time scale. The update in Eq. 11 for single-step TD SARSA (Δ) can be expressed with numerous time-scales by utilizing the principle of splitting the update into several discount factors¹ as demonstrated below:

$$D_m(s_n, a_n) = \mathbb{E} \left[(\eta_m - \eta_{m-1}) Q_{\eta_{m-1}}(s_{n+1}, a_{n+1}) + \eta_m D_m(s_{n+1}, a_{n+1}) \right] \quad (12)$$

In this context, $Q_{\eta_m}(s_n, a_n)$ denotes the action-value function with a discount factor (η_m). $D_m(s_n, a_n)$ denotes the delta function for SARSA (Δ), while η_m represents the discount factor for time-scale m, while $Q_{\eta_m}(s_{n+1}, a_{n+1})$ is the SARSA update for the action-value associated with the subsequent state-action pair. The single-step TD SARSA (Δ) mirrors the original TD update,

but it is applied to action-value functions Q rather than state-value functions V . The single-step Bellman Eq. for the Q -value function in standard SARSA is as follows:

$$Q_{\eta_m}(s_n, a_n) = \mathbb{E}[r_n + \eta_m Q_{\eta_m}(s_{n+1}, a_{n+1})] \quad (13)$$

This denotes the expected value of the reward acquired at time n , in addition to the discounted value of the action-value function in the subsequent state-action pair (s_{n+1}, a_{n+1}) . Currently, instead of utilizing a single discount factor η , we are using a series of discount factors $\eta_0, \eta_1, \dots, \eta_m$. The delta function $D_m(s_n, a_n)$ is defined as the difference between action-value functions associated with successive discount factors from Eq. 9:

$$D_m(s_n, a_n) = Q_{\eta_m}(s_n, a_n) - Q_{\eta_{m-1}}(s_n, a_n) \quad (14)$$

To expand the delta function D_m utilizing its definition, we replace the Bellman Eq. for both Q_{η_m} and $Q_{\eta_{m-1}}$, thereby expressing D_m in relation to these action-value functions. Commence by composing:

$$\begin{aligned} D_m(s_n, a_n) &= Q_{\eta_m}(s_n, a_n) - Q_{\eta_{m-1}}(s_n, a_n) \text{ From Eq. 14} \\ D_m(s_n, a_n) &= \mathbb{E}[r_n + \eta_m Q_{\eta_m}(s_{n+1}, a_{n+1})] - \mathbb{E}[r_n + \eta_{m-1} Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})] \text{ From Eq. 13} \end{aligned}$$

Since the reward terms r_n are included in both formulations, simplify by eliminating them.

$$D_m(s_n, a_n) = \mathbb{E}[\eta_m Q_{\eta_m}(s_{n+1}, a_{n+1}) - \eta_{m-1} Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})]$$

Breaking down the terms facilitates a more thorough examination of the expression. Utilizing the recursive relationship from Eq. 14

$$Q_{\eta_m}(s_{n+1}, a_{n+1}) = D_m(s_{n+1}, a_{n+1}) + Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})$$

Substitute the definition of D_m into the Eq. below.

$$D_m(s_n, a_n) = \mathbb{E}[\eta_m (D_m(s_{n+1}, a_{n+1}) + Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})) - \eta_{m-1} Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})]$$

Make the terms simpler:

$$D_m(s_n, a_n) = \mathbb{E}[\eta_m D_m(s_{n+1}, a_{n+1}) + (\eta_m - \eta_{m-1}) Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})]$$

Integrate the terms to derive the final update Eq. for $D_m(s_n, a_n)$.

$$D_m(s_n, a_n) = \mathbb{E}[(\eta_m - \eta_{m-1}) Q_{\eta_{m-1}}(s_{n+1}, a_{n+1}) + \eta_m D_m(s_{n+1}, a_{n+1})] \quad (15)$$

Eq. 15 shows that the difference in discount factors $\eta_m - \eta_{m-1}$ is integrated into the update for D_m with regard to the action-value function $Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})$ and the bootstrapping from the next step $D_m(s_{n+1}, a_{n+1})$. The Eq. 15 denotes a Bellman Eq. 13 for D_m , combining a decay factor η_m and the reward $Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})$. Consequently, it can be utilized to define the expected TD update for D_m . In this expression, $Q_{\eta_{m-1}}(s_{n+1}, a_{n+1})$ can be expressed as the summation of $D_m(s_{n+1}, a_{n+1})$ for $m \leq M - 1$, indicating that the Bellman Eq. 13 for D_m is contingent upon the values of all delta functions D_m for $m \leq M - 1$. This approach treats the delta value function at each time-scale as an autonomous RL issue, with rewards obtained from the action-value function of the immediately preceding time-scale. Consequently, for a target discounted action-value function $Q_{\eta_m}(s, a)$, all delta components can be trained concurrently through a TD update, employing the prior values of each estimator for bootstrapping. This process necessitates the assumption of a sequence of discount factors, denoted as η_m , which includes both the minimum and maximum values, η_0 and η_m ¹.

4.3 Multi-Step TD (SARSA(Δ))

Numerous studies indicate that multi-step TD methods typically exhibit greater efficiency compared to single-step TD methods¹². In multi-step TD SARSA(Δ), rewards are aggregated over multiple steps instead of depending on a single future reward. This approach takes into account the differences between consecutive discount factors, η_m and η_{m-1} , while utilizing the value function at state s_{n+k_m} for bootstrapping. In SARSA(Δ), the update Eqs. are adjusted to combine rewards over a sequence of transitions before bootstrapping from either the Q -values or delta estimators. In multi-step SARSA(Δ), the agent accumulates rewards over K steps, resulting in a necessary adjustment to the update rule for each D_m .

$$\begin{aligned} D_m(s_n, a_n) &= \mathbb{E} \left[\sum_{x=1}^{k_m-1} (\eta_m^x - \eta_{m-1}^x) r_{n+x} + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) Q_{\eta_{m-1}}(s_{n+k_m}, a_{n+k_m}) \right. \\ &\quad \left. + \eta_m^{k_m} D_m(s_{n+k_m}, a_{n+k_m}) \right] \end{aligned} \quad (16)$$

In this case, r_{n+x} represents the reward obtained at time step $n+x$. The term a_{n+k_m} and the action are selected in a greedy manner as $a = \arg \max_a Q(s_{n+k_m}, a)$, with k_m representing the number of steps associated with the discount factor η_m . This approach extends Q-learning to multi-step temporal difference learning by gathering rewards over multiple stages and employing bootstrapping from both the present and previous time-scale action-value functions. In normal multi-step TD learning, the TD error is computed by aggregating rewards over several stages rather than relying on a single future step, followed by bootstrapping from the action-value at the last step. For instance, employing a single discount factor η , the multi-step TD update Eq. can be articulated as:

$$Q(s_n, a_n) = \mathbb{E} \left[\sum_{x=0}^{k-1} \eta^x r_{n+x} + \eta^k Q(s_{n+k}, a_{n+k}) \right] \quad (17)$$

Herein, the initial segment of the Eq. aggregates the rewards across k steps, each diminished by η . The second component derives from the action-value function at time step $n+k$. We expand this methodology to include various discount factors and delta decomposition, concentrating on representing the delta function $D_m(s_n, a_n)$ as the difference between action-value functions with discount factors η_m and η_{m-1} . According to the definition of D_m , we obtain:

$$D_m(s_n, a_n) = Q_{\eta_m}(s_n, a_n) - Q_{\eta_{m-1}}(s_n, a_n)$$

The multi-step variant of the Bellman Eq. is now implemented for both $Q_{\eta_m}(s_n, a_n)$ and $Q_{\eta_{m-1}}(s_n, a_n)$. The multi-step Bellman Eq. for $Q_{\eta_m}(s_n, a_n)$ is expressed as follows:

$$Q_{\eta_m}(s_n, a_n) = \mathbb{E} \left[\sum_{x=0}^{k_m-1} \eta_m^x r_{n+x} + \eta_m^{k_m} Q_{\eta_m}(s_{n+k_m}, a_{n+k_m}) \right]$$

For $Q_{\eta_{m-1}}(s_n, a_n)$, the multi-step Bellman Eq. is expressed as follows:

$$Q_{\eta_{m-1}}(s_n, a_n) = \mathbb{E} \left[\sum_{x=0}^{k_m-1} \eta_{m-1}^x r_{n+x} + \eta_{m-1}^{k_m} Q_{\eta_{m-1}}(s_{n+k_m}, a_{n+k_m}) \right]$$

The formulas for the delta estimator are utilized to perform the subtraction of the two Eqs.

$$D_m(s_n, a_n) = \left[\sum_{x=0}^{k_m-1} \eta_m^x r_{n+x} + \eta_m^{k_m} Q_{\eta_m}(s_{n+k_m}, a_{n+k_m}) \right] - \left[\sum_{x=0}^{k_m-1} \eta_{m-1}^x r_{n+x} + \eta_{m-1}^{k_m} Q_{\eta_{m-1}}(s_{n+k_m}, a_{n+k_m}) \right]$$

The expressions are broadened and simplified. The immediate reward terms r_{n+x} are present in both Eqs. but are adjusted by distinct discount factors, η_m and η_{m-1} . This enables us to express the disparity in rewards as:

$$\sum_{x=0}^{k_m-1} \eta_m^x r_{n+x} - \sum_{x=0}^{k_m-1} \eta_{m-1}^x r_{n+x} = \sum_{x=0}^{k_m-1} (\eta_m^x - \eta_{m-1}^x) r_{n+x}$$

Subsequently, for the bootstrapping terms, the recursive relationship is utilized as outlined:

$$Q_{\eta_m}(s_{n+k_m}, a_{n+k_m}) = D_m(s_{n+k_m}, a_{n+k_m}) + Q_{\eta_{m-1}}(s_{n+k_m}, a_{n+k_m})$$

As a result, the following expression is obtained:

$$\eta_m^{k_m} Q_{\eta_m}(s_{n+k_m}, a_{n+k_m}) - \eta_{m-1}^{k_m} Q_{\eta_{m-1}}(s_{n+k_m}, a_{n+k_m}) = (\eta_m^{k_m} - \eta_{m-1}^{k_m}) Q_{\eta_{m-1}}(s_{n+k_m}, a_{n+k_m}) + \eta_m^{k_m} D_m(s_{n+k_m}, a_{n+k_m})$$

The definitions for both the rewards and bootstrapping are now integrated to yield the final expression:

$$D_m(s_n, a_n) = \mathbb{E} \left[\sum_{x=0}^{k_m-1} (\eta_m^x - \eta_{m-1}^x) r_{n+x} + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) Q_{\eta_{m-1}}(s_{n+k_m}, a_{n+k_m}) + \eta_m^{k_m} D_m(s_{n+k_m}, a_{n+k_m}) \right] \quad (18)$$

Eq. 18 extends the conventional multi-step TD update to include various multiple discount factors and action-value functions in RL. As a result, each D_m obtains a share of the rewards from the environment up to time-step $k_m - 1$. Furthermore, each D_m utilizes its distinct action-value function in conjunction with the value from the prior time scale. A variation of this algorithm, utilizing k -step bootstrapping as described in¹², is presented in Algorithm 1. Despite Algorithm 1 exhibiting quadratic complexity in relation to M , it can be executed with linear complexity for substantial M by preserving \hat{Q} action-values at each time-scale η_m .

Algorithm 1 Multi-Step TD (SARSA(Δ))

Inputs: Pick out the discount factors ($\eta_0, \eta_1, \dots, \eta_M$), bootstrapping steps (k_0, k_1, \dots, k_M), and learning rates ($\alpha_0, \alpha_1, \dots, \alpha_M$). Set the initial value of $D_m(s, a) = 0$ for all states, actions, and scales m .

```
for episode = 0, 1, 2,... do                                     ▷ Loop for each episode
  Initialize state  $s_0$  and choose an initial action  $a_0$  in accordance with a policy.
  for n = 0, 1, 2,... do                                         ▷ Loop for each time step
    Take action  $a_n$ , observe reward  $r_n$  and next state  $s_{n+1}$ .
    Select action  $a_{n+1}$  based on a policy.
    for m = 0, 1, ..., M do
      if m = 0 then
         $G^0 = \sum_{x=0}^{k_0-1} \eta_0^x r_{n+x} + \eta_0^{k_0} D_0(s_{n+k_0}, a_{n+k_0})$ 
      else ▷ Utilizing Eq. 10, we substitute  $Q_{\eta_{m-1}}(s_{n+k_m}, a_{n+k_m})$  by summing the D-components up to  $D_{m-1}$  in Eq. 18.
         $G^m = \sum_{x=0}^{k_m-1} (\eta_m^x - \eta_{m-1}^x) r_{n+x} + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) \sum_{m=0}^{m-1} D_{m-1}(s_{n+k_m}, a_{n+k_m}) + \eta_m^{k_m} D_m(s_{n+k_m}, a_{n+k_m})$ 
      end if
    end for
    for m = 0, 1, 2,..., M do
       $D_m(s_n, a_n) \leftarrow D_m(s_n, a_n) + \alpha_m (G^m - D_m(s_n, a_n))$ 
    end for
  end for
end for
```

4.4 SARSA TD(λ, Δ)

Eq. 19 presents the λ -return^{2,34}, which integrates rewards across several steps to establish a target for TD(λ) updates. The λ -return $G_n^{\eta, \lambda}$ is formally defined as follows:

$$G_n^{\eta, \lambda}(s_n, a_n) = \hat{Q}_\eta(s_n, a_n) + \sum_{k=0}^{\infty} (\lambda \eta)^k \delta_{n+k}^\eta \quad (19)$$

Eq. 20 defines the TD(λ) operator, which is used to iteratively apply λ -returns in updating the value functions. For SARSA, the TD(λ) operator updates the action-value function by summing the λ -discounted TD errors as follows:

$$T_\lambda Q(s_n, a_n) = Q(s_n, a_n) + (I - \lambda \eta P)^{-1} (TQ(s_n, a_n) - Q(s_n, a_n)) \quad (20)$$

P represents the transition matrix for state-action pairs according to the policy π , while Q denotes the action-value function. Similarly, Eq. 21 defines the λ -return specific to delta estimators in TD(λ, Δ), denoted as G_n^{m, λ_m} for each delta estimator D_m :

$$G_n^{m, \lambda_m} := \hat{D}_m(s_n, a_n) + \sum_{k=0}^{\infty} (\lambda_m \eta_m)^k \delta_{n+k}^m \quad (21)$$

where $\delta_n^0 := \delta_n^{\eta_0}$ and $\delta_n^m := (\eta_m - \eta_{m-1}) \hat{Q}_{\eta_{m-1}}(s_{n+1}, a_{n+1}) + \eta_m \hat{D}_m(s_{n+1}, a_{n+1}) - \hat{D}_m(s_{n+1}, a_{n+1})$ are the TD-errors.

4.5 SARSA TD(λ, Δ) with Generalized Advantage Estimation (GAE)

Generalized Advantage Estimation (GAE)³⁹ seeks to compute the advantage function by aggregating multi-step TD errors. To apply this approach to delta estimators $D_m(s_n, a_n)$, advantage estimates $A^\Delta(s_n, a_n)$ are computed for each time scale. Each advantage estimate $A^\Delta(s_n, a_n)$ employs a multi-step TD error pertinent to the delta estimator D_n , expressed as follows:

$$A^\Delta(s_n, a_n) = \sum_{k=0}^{T-1} (\lambda_m \eta_m)^k \delta_{n+k}^\Delta \quad (22)$$

where $\delta_{n+k}^\Delta := r_n + \eta_m \sum_{m=0}^M \hat{D}_m(s_{n+1}, a_{n+1}) - \sum_{m=0}^M \hat{D}_m(s_n, a_n)$.

The discount factor η_m is utilized, and the sum of all D estimators as a surrogate for Q_{η_m} . This objective is applicable to PPO by implementing the policy update from Eq. 7 and replacing A with A^Δ . Additionally, to train each D_m , a truncated form of their corresponding λ -return are used, as outlined in Eq. 21. For more information, see Algorithm 2.

Algorithm 2 PPO-TD(λ , SARSA(Δ))

Inputs: Pick out the discount factors ($\eta_0, \eta_1, \dots, \eta_M$), bootstrapping steps (k_0, k_1, \dots, k_M), and learning rates ($\alpha_0, \alpha_1, \dots, \alpha_M$). Set the initial value of $D_m(s, a) = 0$ for all states s , actions a , and scales m .

Initialize policy v , and values $\theta^m \forall m$

for episode = 0, 1, 2, ... **do**

▷ Loop for each episode

Initialize state s_0 and choose an initial action a_0 in accordance with a policy.

for $n = 0, 1, 2, \dots$ **do**

▷ Loop for each time step

Take action a_n , observe reward r_n and next state s_{n+1} .

Select next action a_{n+1} based on a policy.

for $m = 0, 1, \dots, M$ **do**

if $n \geq T$ **then**

$G^{m, \lambda_m} \leftarrow \hat{D}_m(s_{n-T}, a_{n-T}) + \sum_{k=0}^{T-1} (\lambda_m \eta_m)^k \delta_{n-T+k}^m \forall m$ ▷ Computing multi-step return G^{m, λ_m} and TD-error δ_{n-T+k}^m using Eq. 21.

end if

end for

for $m = 0, 1, 2, \dots, M$ **do**

$\hat{D}_m(s_{n-T}, a_{n-T}) \leftarrow \hat{D}_m(s_{n-T}, a_{n-T}) + \alpha_m (G^{m, \lambda_m} - \hat{D}_m(s_{n-T}, a_{n-T}))$

$A^\Delta = \sum_{k=0}^{T-1} (\lambda_m \eta_m)^k \delta_{n-T+k}^\Delta$ ▷ Where A^Δ and δ_{n-T+k}^Δ are computed using Eq. 22.

$\theta^m \leftarrow \theta^m + \alpha_m (G^{m, \lambda_m} - \hat{D}_m(s_{n-T}, a_{n-T})) \nabla \hat{D}_m(s_{n-T}, a_{n-T})$ ▷ Update θ^m with TD (Eq. 3) using $G^{m, \lambda_m} \forall m$.

$\mathcal{L}(v) = \mathbb{E} \left[\min \left(\rho(v) A^\Delta(s, a), \text{clip}(\rho(v), 1 - \epsilon, 1 + \epsilon) A^\Delta(s, a) \right) \right]$ ▷ from Eq. 7

$v \leftarrow v + \alpha_v \nabla_v \mathcal{L}(v)$ ▷ Update the policy parameters v with PPO (Eq. 7) for SARSA using A^Δ

end for

end for

end for

5 Analysis

Subsequently, the delta estimators are analyzed in relation to the bias-variance trade-off. In SARSA, bias arises from reliance on current estimates, influencing updates over short time scales, especially with smaller discount factors, like lower values of η in D_m . Utilizing smaller discount factors in updates leads to a decrease in variance because they depend on current estimates. However, this approach also results in heightened bias, as it does not incorporate significant information about future rewards. Conversely, variance increases for updates linked to longer time scales, as future reward information introduces additional variability. In TD(Δ), action-value components D_m with higher η_m are associated with greater long-term rewards; however, this results in increased variance stemming from the stochastic characteristics of reward sequences. We begin by demonstrating that our estimator is equivalent to the standard estimator \hat{Q}_η under specific conditions, as outlined in Theorem 1. This comparison elucidates the essential metrics of our estimator that may indicate potential advantages over the standard \hat{Q}_η estimator. Building on this result and previous research by Kearns and Singh¹¹, the analyses are adapted for SARSA to investigate the effects of bias and variance, along with the TD(Δ) decomposition framework, in relation to action-value functions. Our objective is to extend the bias-variance error bound framework to action-value settings using TD(Δ)(i.e., Theorem 4), thereby offering a better understanding of how these quantities can be balanced to obtain optimal results¹.

5.1 SARSA Equivalency Configurations and Enhancements

In certain scenarios, we can demonstrate that our delta estimators for SARSA correspond with the traditional action-value estimator when reformulated into an action-value function. This discourse focuses on the approximation of the linear function of the specified form:

$$\hat{Q}(s, a)_\eta := \langle \theta^\eta, \phi(s, a) \rangle \quad \text{and} \quad \hat{D}_m(s, a) := \langle \theta^m, \phi(s, a) \rangle, \forall m$$

where θ and θ^m represent weight vectors in \mathbb{R}^d and the function $\phi : S \times A \rightarrow \mathbb{R}^d$ represents a mapping from a state-action pair to a specified d-dimensional feature space. The weight vector θ for SARSA is updated according to the TD(λ) learning rule in the following manner:

$$\theta_{n+1}^\eta = \theta_n^\eta + \alpha \left(G_n^{\eta, \lambda} - \hat{Q}_\eta(s, a) \right) \phi(s_n, a_n), \quad (23)$$

Here, $G_n^{\eta, \lambda}$ denotes the $TD(\lambda)$ return as stated in Eq. 19. Likewise, the delta estimator approach $TD(\lambda_m, \Delta)$ is used to update each \hat{D}_m :

$$\theta_{n+1}^m = \theta_n^m + \alpha \left(G_n^{M, \lambda_m} - \hat{D}_m(s_n, a_n) \right) \phi(s_n, a_n), \quad (24)$$

where G_n^{M, λ_m} is specified identically to the $TD(\Delta)$ return defined in Eq. 21, modified for the particular action-value function. In these equations, α and $\{\alpha_m\}_m$ represent positive learning rates. Two SARSA version algorithms are shown to be equivalent in this context by the following theorem. The subsequent theorems resemble those presented by Romoff et al.¹; however, a proof is presented for the action-value function in relation to SARSA, whereas Romoff et al. established the proof for the value function.

Theorem 1. *Inspired by Romoff et al.¹, if $\alpha_m = \alpha$, $\lambda_m \eta_m = \lambda \eta$, $\forall m$, and if we choose the initial conditions in such a way that $\sum_{m=0}^M \theta_0^m = \theta_0^\eta$, then the iterates produced by $TD(\lambda)$ (Eq. 23) and $TD(\lambda, \Delta)$ (Eq. 24) with linear function approximation satisfy the following conditions:*

$$\sum_{m=0}^M \theta_n^m = \theta_n^\eta \quad (25)$$

The proof is presented in appendix A.

Equivalence is attained when $\lambda_m \eta_m = \lambda \eta$, $\forall m$. When λ approaches 1 and η_m is less than η , this condition suggests that $\lambda_m = \lambda \eta / \eta_m$ could potentially surpass one, resulting in a risk of divergence in $TD(\lambda_m)$. The subsequent theorem demonstrates that the $TD(\lambda)$ operator, as defined in Eq. 20, qualifies as a contraction mapping for the range $1 \leq \lambda < \frac{1+\eta}{2\eta}$, thereby confirming that $\lambda \eta < 1$.

Theorem 2. *Inspired by Romoff et al.¹, $\forall \lambda \in \left[0, \frac{1+\eta}{2\eta}\right]$, the operator T_λ is defined by the Eq. $T_\lambda Q = Q + (I - \lambda \eta P)^{-1} (TQ - Q)$, $\forall Q \in \mathbb{R}^{|S| \times |A|}$, is well-defined. Furthermore, $T_\lambda Q$ constitutes a contraction in relation to the max norm, with its contraction coefficient expressed as $\frac{\eta}{|1 - \lambda \eta|}$.*

The proof is presented in appendix A.

The analysis presented in Theorem 1 can be adapted to a new context in which k_m -step $TD(\Delta)$ is employed for each D_m component, rather than $TD(\lambda, \Delta)$, a variant of TD learning. Theorem 1 demonstrates that with linear function approximation, standard multi-step TD and multi-step $TD(\Delta)$ can be equivalent when the number of steps (k_m) is consistent across all time scales (i.e., $k_m = k$, $\forall m$).

Although these methods could theoretically be equivalent, such equivalence is not common. To maintain the preservation of equivalence, it is crucial for the learning rate to remain consistent across all time scales. This underscores a significant limitation, as shorter time scales, which involve fewer future steps, can be acquired more rapidly than longer time scales that require the assessment of additional steps beforehand. Furthermore, in practical applications, particularly involving nonlinear function approximation such as deep neural networks, adaptive optimizers are frequently employed^{39,40}. The optimizers modify the learning rate accordance with the complexity of the learning task, which depends on the attributes of the delta estimator and its target. There is no universally applicable learning rate in this setting since the effective learning rate changes with time scale. In addition to the learning rate, breaking the action-value function into separate components (especially the D_m components) has advantages that standard TD learning, such as non-delta estimators, does not have. The ability to utilize various k -step returns (or λ -return) across multiple time scales enhances control and adaptability in the learning process. If $k_m < k_{m+1}$, $\forall m$ (or $\eta_m \lambda_m < \eta_{m+1} \lambda_{m+1}$, $\forall m$), this approach has the potential to reduce variance while simultaneously introducing bias, since shorter time scales, which can be learned more quickly, may not correspond perfectly with longer time scales regarding the values they estimate.

5.2 Evaluation for Minimizing k_m Values in SARSA via Phased Updates

To show how our approach differs from the single estimator case, let's follow the tabular phased version of k -step TD presented by Kearns and Singh¹¹. In SARSA, the goal is to evaluate the value associated with a state-action pair (s, a) starting from each state $s \in \mathcal{S}$ and with action $a \in \mathcal{A}$, then n trajectories are generated $\left\{ S_0^{(x)} = s, a_0, \eta_0, \dots, S_k^{(x)}, a_k^{(x)}, \eta_k^{(x)}, S_{k+1}^{(x)}, \dots \right\}_{1 \leq x \leq n}$ by following the policy π and averaging over trajectories. Let $Q_{\eta, n}(s, a)$ denote the Q -value estimate at phase n for the discount factor η . For each iteration n , also called phase n , obtain the phase-based estimate $Q_{\eta, n}(s, a)$ for (s, a) ; we average over the trajectories, resulting in:

$$\hat{Q}_{\eta, n}(s, a) = \frac{1}{n} \sum_{x=1}^n \left(\sum_{i=0}^{k-1} \gamma^i r_i^{(x)} + \eta^k \hat{Q}_{\eta, n-1}(s_k^{(x)}, a_k^{(x)}) \right) \quad (26)$$

Theorem 3 draws upon the findings of Kearns and Singh¹¹, who demonstrated the result concerning the state-value function. In contrast, the subsequent theorem introduces an upper limit on the error associated with action-value function estimation, expressed as $\Delta_n^{\hat{Q}_\eta} = \max_{s,a} \left\{ \left| \hat{Q}_{\eta,n}(s,a) - Q_\eta(s,a) \right| \right\}$

Theorem 3. Inspired by Kearns and Singh¹¹, for any $0 < \delta < 1$, let $\epsilon = \sqrt{\frac{2\log(2k/\delta)}{n}}$. with probability $1 - \delta$,

$$\Delta_n^{\hat{Q}_\eta} \leq \underbrace{\epsilon \left(\frac{1 - \eta^k}{1 - \eta} \right)}_{\text{variance term}} + \underbrace{\eta^k \Delta_{n-1}^{\hat{Q}_\eta}}_{\text{bias term}} \quad (27)$$

The proof is presented in appendix A.

In Eq. 27, a variance term $(\frac{1-\eta^k}{1-\eta})$ is arising due to sampling error from rewards collected along trajectories. In particular, ϵ bounds the deviation of the empirical average of rewards from the true expected reward. The second term $(\eta^k \Delta_{n-1}^{\hat{Q}_\eta})$ in Eq. 27 is a bias term that arises from bootstrapping off the previous phase's estimates.

Similarly, a phased SARSA variant of the multi-step $TD(\Delta)$ method is considered. For each phase n , each D component is updated as follows:

$$\hat{D}_{m,n}(s,a) = \frac{1}{n} \sum_{x=1}^n \left(\sum_{i=1}^{k-1} (\eta_m^i - \eta_{m-1}^i) r_i^{(x)} + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) Q_{\eta_{m-1}}(s_{n+k}^{(x)}, a_{n+k}^{(x)}) + \eta_m^{k_m} \hat{D}_m(s_{n+k}^{(x)}, a_{n+k}^{(x)}) \right) \quad (28)$$

The maximum threshold for the phased error has now been established. The cumulative errors of each D component, denoted as $TD(\Delta)$, is expressed as $\sum_{m=0}^M \Delta_n^m$, where $\Delta_n^m = \max_{s,a} \left\{ \left| \hat{D}_m(s,a) - D_m(s,a) \right| \right\}$. Essential differences between SARSA and Q-learning regarding the error bound: In SARSA, the error is defined by the actions taken by the agent, indicating that the Q-values are modified based on the state-action pairings encountered by the agent during exploration. The inaccuracy in Q-learning emerges from the greedy action, which is defined as the action that maximizes the Q-value for the future state, assuming that the agent consistently operates in an optimal manner.

Theorem 4. Inspired by Romoff et al.¹, assume that $\eta_0 \leq \eta_1 \leq \eta_2 \leq \dots \leq \eta_m = \eta$ and $K_0 \leq K_1 \leq \dots \leq K_m = K$, for any $0 < \delta < 1$, let $\epsilon = \sqrt{\frac{2\log(2k/\delta)}{n}}$, with probability $1 - \delta$,

$$\sum_{m=0}^M \Delta_n^m \leq \underbrace{\epsilon \left(\frac{1 - \eta^k}{1 - \eta} \right)}_{\text{variance reduction}} + \underbrace{\epsilon \left(\sum_{m=0}^{M-1} \frac{\eta_m^{k_{m+1}} - \eta_m^{k_m}}{1 - \eta_m} \right)}_{\text{bias introduction}} + \sum_{m=0}^{M-1} \left(\eta_m^{k_m} - \eta_m^{k_{m+1}} \right) \sum_{q=0}^m \Delta_{n-1}^q + \eta^k \sum_{m=0}^M \Delta_{n-1}^m \quad (29)$$

The proof is presented in appendix A.

The authors examine the comparison of the limits for phased $TD(\lambda)$ in theorem 3 with those for phased $TD(\Delta)$ in theorem 4. This comparison demonstrates that phased $TD(\Delta)$ facilitates variance reduction equivalent to $\epsilon \sum_{m=0}^{M-1} \frac{\eta_m^{k_{m+1}} - \eta_m^{k_m}}{1 - \eta_m} \leq 0$ but introduces a potential bias quantified as $\sum_{m=0}^{M-1} \left(\eta_m^{k_m} - \eta_m^{k_{m+1}} \right) \sum_{q=0}^m \Delta_{n-1}^q \geq 0$. Utilizing phased $TD(\Delta)$ to decrease k_m synchronizes updates more effectively with recent actions, hence lowering variance from high-discounted return components. Shortened k_m values accrue bias more swiftly, as SARSA bootstraps from current policy estimations that may vary over phases. It is observed that when all k_m values are identical, both algorithms yield the same upper bound.

In SARSA, the return estimate includes subsequent states and actions determined by the policy. The anticipated discounted return over T steps closely approximates the infinite-horizon discounted expected return after T , where $T \approx \frac{1}{1-\eta}$ ⁴¹. Consequently, k_m can be effectively simplified for any η_m such that $k_m \approx \frac{1}{1-\eta_m}$, thereby adhering to this principle. Therefore, with T samples, similar to $TD(\Delta)$ ¹, SARSA employing $TD(\Delta)$ utilizes $k_m \approx \frac{1}{1-\eta_m}$ to establish a suitable balance between bias and variance over all time scales significantly less than T . By establishing k_m for each η_m , where $\eta_m^{\frac{1}{1-\eta_m}} \leq \frac{1}{e}$, the approach guarantees that the effective horizon and variance are maintained within limits without requiring extensive parameter optimization. Each increase in D_m facilitates the doubling of the effective horizon, hence ensuring logarithmic scaling with the quantity of action-value functions. SARSA can utilize this to modify additional parameters in alignment with the specified time scales, enhancing the algorithm's efficiency in calculating long-term rewards without added complexity.

6 Experiments and Results

Experiments were conducted in three distinct environments to evaluate the effectiveness of the proposed method: the Tabular Ring MDP environment detailed in Section 6.1, the deterministic and stochastic OpenAI Gym environments examined in Section 6.2, and the Atari environments from OpenAI Gym described in Section 6.3. This section initiates with a detailed overview of the experimental setup.

Experimental Setup: Experiments were conducted using OpenAI Gym control tasks⁴², and Figs.1, 3, and 6 show the relevant environments. The experiments were carried out on a single PC with 16 GB of memory, an Intel Core i7-2600 processor, and a GPU. The machine ran on 64-bit Ubuntu 22.04.5 LTS. Python 3.9 was used during development, along with the PyTorch package. In each experiment, ten randomly selected seeds were utilized, and the average results are provided. The remaining hyperparameter settings are displayed in the table 4 of appendix B.

6.1 Tabular

Q-value iteration is utilized to determine the optimal Q-value function. An open-source implementation of Value Iteration (VI), based on the code by Peter Henderson⁴³, is employed in accordance with the algorithmic principles outlined in Bellman³³ and Sutton et al.¹². The methodology calculates the optimal value function; however, this approach is adjusted to ascertain the optimal Q-value function. Upon acquiring the optimal Q-values, Temporal Difference (TD) and SARSA(Δ) methods are applied to derive approximation Q-values via sampling. After 5000 steps, each learning iteration comes to an end. For every setup, 200 trials are executed using different random seeds that influence the dynamics of the transition. The algorithms are implemented using the codebase provided by Kostrikov⁴⁴.

The proposed method is contrasted with k-step SARSA, as described in Sutton et al. [12, Chapter 7.2]. SARSA (Δ) is evaluated against different RL benchmarks, including the Ring MDP¹¹ and both deterministic and stochastic environments, to determine its efficacy. Fig. 1 illustrates the Ring MDP, which consists of a Markov process featuring five states arranged in a circular form. The probability of transitioning one state clockwise around the ring is 0.95, while the likelihood of remaining in the current state at each step is 0.05. In contrast to the other states on the ring that provide a reward of 0, two neighboring states present rewards of +1 and -1, respectively. All experiments utilize the identical parameters established by Romoff et al.¹, which

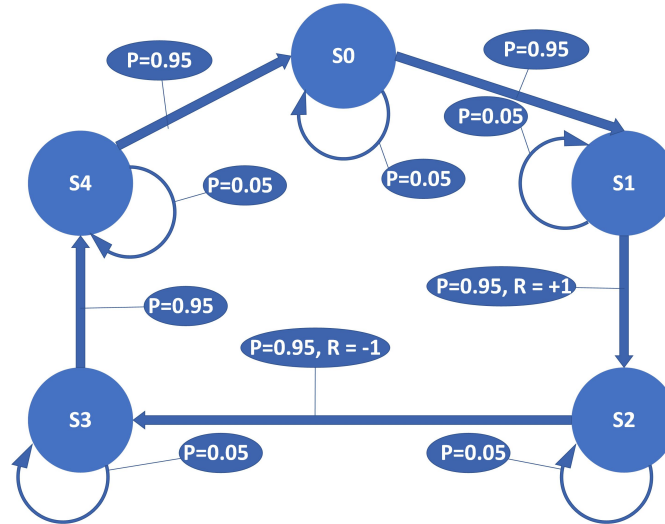


Figure 1. Five-state ring MDP as described by Kearns and Singh¹¹

involve supplying a variable number of gamma values, commencing at 0 and incrementing according to $\eta_{m+1} = \frac{\eta_m + 1}{2}$ until the requisite maximum η_m is attained. As previously mentioned in Section 5.2, $K_m = \frac{1}{(1-\eta_m)}$ for every m. The baseline consists of a singular estimator with $\eta = \eta_m$ and $k = k_m$. Fig. 2a illustrates the effectiveness of two methodologies: K-step SARSA (Δ) and K-step SARSA TD, utilizing a discount factor ($\eta = 0.996$), $k = 64$ (a parameter related to the algorithm's step count), and various values of the learning rate α . The y-axis represents the absolute error, while the x-axis indicates the parameter α , which ranges from 0.0 to 1.0. At $\alpha=0.0$, both methods start with about the same amount of error (around 0.00230). As α increases, K-step SARSA (Δ) exhibits a slight initial reduction in inaccuracy, followed by a consistent rise thereafter. In contrast, K-step SARSA TD shows a constant rise in error with low variability. Overall, the K-step SARSA (Δ) approach outperforms K-step SARSA TD in terms of stability and accuracy across all α values. Both techniques are sensitive to fluctuations in α , but K-step

SARSA TD has a more significant rise in error.

Fig. 2b compares the efficacy of two approaches, K-step SARSA (Δ) and K-step SARSA TD, based on the number of k-steps. The y-axis represents the absolute error, whereas the x-axis depicts k-steps, which relates to the number of steps utilized in the algorithms. The range of k-steps spans from 0 to approximately 250. K-step SARSA (Δ) begins with a lower error at smaller k-steps, which then escalates with larger k-steps. K-step SARSA TD starts with an error level similar to that of K-step SARSA (Δ) for lower k-steps; however, its error increases more sharply as k-steps increase. The overall pattern indicates that both techniques exhibit a rise in inaccuracy as the number of k-steps increases. K-step SARSA (Δ) consistently demonstrates superior performance compared to K-step SARSA TD across all k-step values, evidenced by a lower error rate. The rapid error amplification noted in K-step SARSA TD indicates that the scheduling method employed in K-step SARSA (Δ) successfully mitigates error accumulation. Additionally, in this particular MDP, the error increases with k as a result of the bias-variance trade-off linked to k-step returns, consistent with the findings of Romoff et al.¹, Kearns and Singh¹¹.

Analysis of Figs. 2a and 2b: In comparing the two figures, K-step SARSA (Δ) consistently outperforms K-step SARSA TD in both scenarios (α and k-steps). Both approaches demonstrate sensitivity to increased parameter values (α or k-steps). K-step SARSA (Δ) demonstrates improved stability and achieves lower error rates due to its scheduling methodology. Interestingly, compared to the growth in α , errors climb faster with K-step SARSA TD as the number of k-steps increases.

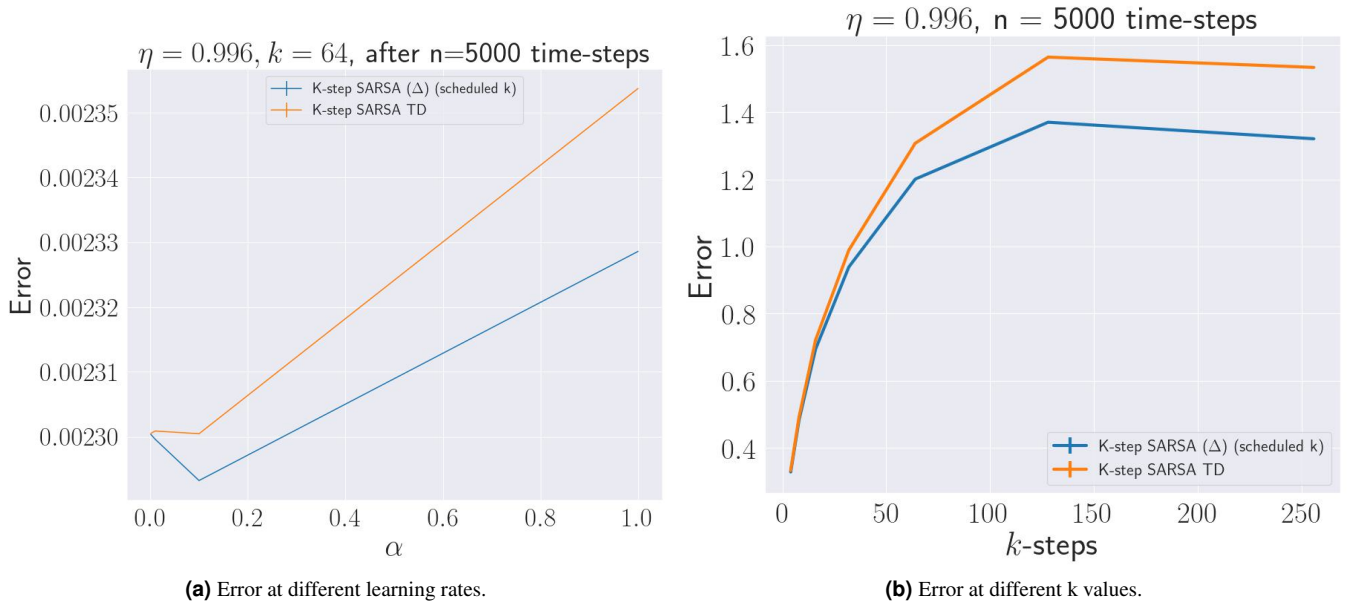


Figure 2. Fig. 2a illustrates the outcomes for $\eta_M = 0.996$ on the five-state Ring MDP. The error signifies the absolute discrepancy between the estimated and actual discounted Q-value function (precomputed via Q-value iteration), averaged across the complete learning trajectory of 5000 timesteps. Fig. 2b illustrates the average absolute error for the optimal learning rate at each k-step return, extending beyond the effective planning horizon of η_M .

6.2 OpenAI Gym Environments

Two algorithms, k-step SARSA(Δ) and k-step SARSA TD, were tested in the FrozenLake-v1, Taxi-v3, CliffWalking-v0, and Blackjack environments of OpenAI Gym. The results are shown in the figures below. The y-axis displays how many awards the algorithms got on average across episodes, while the x-axis shows how many episodes it took to train the algorithms. Episodes range from 0 to 100,000 episodes. The shaded area around each curve displays the standard error for 10 random seeds. Both algorithms utilize $k = 32$, which is the number of steps.

The red line denotes K-step SARSA (Δ), while the blue line represents K-step SARSA TD in both Fig. 4a and Fig. 4b. Fig. 4a presents the FrozenLake-v1 environment, while Fig. 4b showcases the Taxi-v3 environment. Fig. 4a illustrates that k-step SARSA (Δ) consistently achieves higher average rewards compared to k-SARSA TD as training progresses. Over the first 20,000 episodes, K-step SARSA (Δ) shows fast learning with a significant rise in average rewards. However, k-step SARSA TD displays a smaller increase in average rewards when compared to k-step SARSA (Δ) during the same episodes. After 40,000 episodes, k-step SARSA (Δ) performance levels off and regularly beats k-step SARSA TD, showing that it is reliable over the long-term. k-step SARSA consistently performs worse than k-step SARSA (Δ).

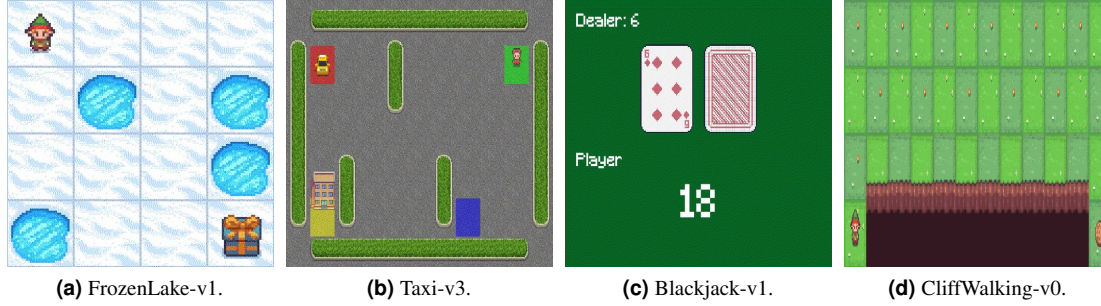


Figure 3. OpenAI Gym control used for all experiments. In order from left to right: Frozen Lake, Taxi, Blackjack, and Cliff Walking.

Fig. 4b illustrates that K-Step SARSA (Δ) regularly achieves superior rewards (i.e., nearer to -200) during the training phase, signifying enhanced learning efficiency and overall performance. Fig. 4b also demonstrates that K-Step SARSA (Δ) stabilizes in the initial phases, indicating robustness and the capacity to converge to an effective policy, whereas K-Step SARSA TD commences at considerably lower rewards (approximately -1800) and exhibits negligible improvement, remaining predominantly flat throughout the training process. The K-Step SARSA TD algorithm encounters difficulties in formulating an effective policy in this context relative to K-Step SARSA (Δ). The results of the experiment indicate that K-Step SARSA (Δ) demonstrates a significant advantage over K-Step SARSA TD in the Taxi-v3 environment, attaining considerably higher average rewards and achieving faster convergence.

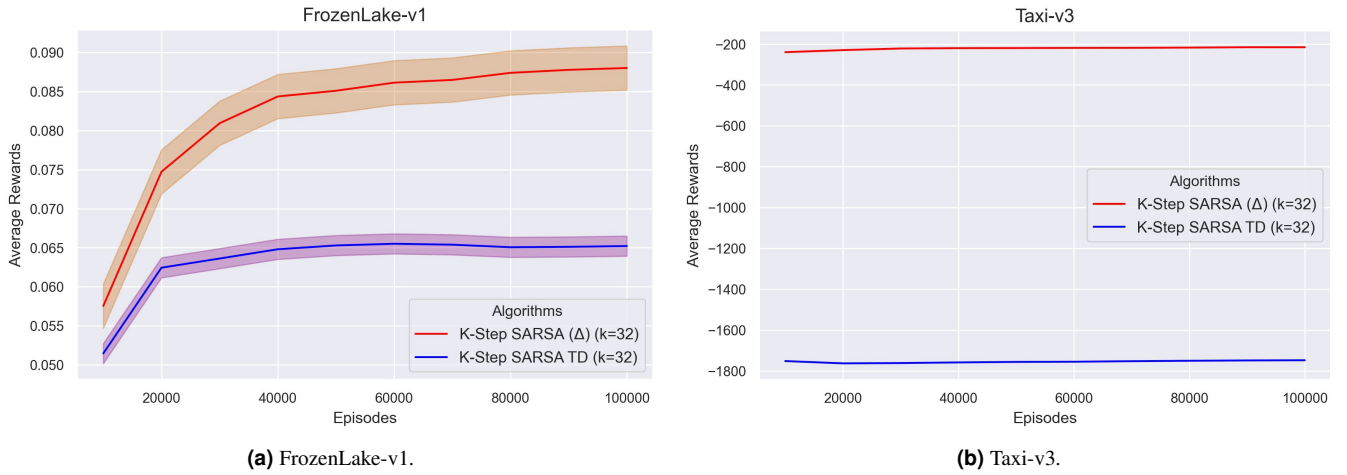


Figure 4. Comparative analysis of K-step SARSA (Δ) versus traditional K-step SARSA TD within the OpenAI Gym environments: FrozenLake-v1 and Taxi-v3. The shaded area illustrates the standard error calculated from 10 random seeds.

Figs. 5a and 5b demonstrate the learning effectiveness of two variations of the K-Step SARSA algorithm: K-Step SARSA (Δ) and K-Step SARSA TD. This analysis is based on 100,000 episodes conducted in the CliffWalking-v0 and Blackjack-v1 environments. The evaluation of both algorithms is conducted with $k=32$, which is a hyperparameter indicating the number of steps involved in the updating process. The overall pattern of average rewards shown in Fig. 5a indicates that both algorithms exhibit an upward trajectory in average rewards as the number of episodes increases, reflecting enhanced policy performance through ongoing learning. Nonetheless, K-Step SARSA (Δ) routinely exhibits better performance than K-Step SARSA TD, leading to increased average rewards across the episodes.

In comparison to K-Step SARSA TD, the red line in Fig. 5a representing K-Step SARSA (Δ) shows a significant improvement in the first episodes (about 20,000 episodes), suggesting a faster convergence to optimum policies. K-Step SARSA (Δ) receives a reward close to -1050 in the 100,000th episode, whereas K-Step SARSA TD receives a somewhat smaller reward of around -1120. This illustrates how K-Step SARSA (Δ) performs better on this task.

The average reward of approximately +0.15 is dependably achieved across all episodes by the red curve in Fig. 5b that

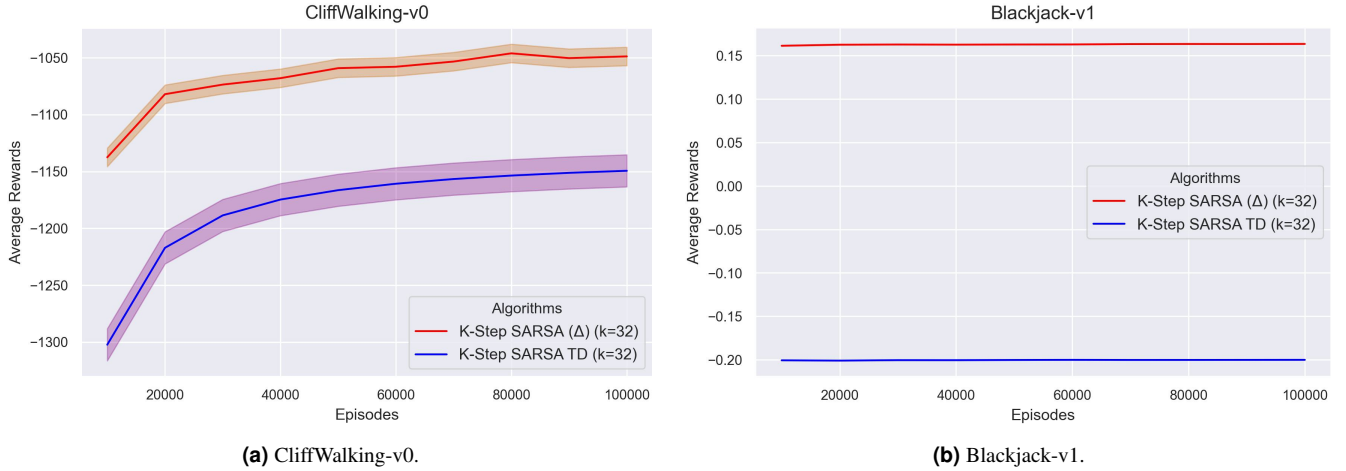


Figure 5. Comparative analysis of K-step SARSA (Δ) versus traditional K-step SARSA TD within the OpenAI Gym environments: CliffWalking-v0 and Blackjack-v1. The shaded area illustrates the standard error calculated from 10 random seeds.

represents K-Step SARSA (Δ). The blue curve in Fig. 5b, which represents K-Step SARSA TD, suggests stability; however, it presents an average reward that is significantly lower, at approximately -0.2, across all episodes. K-Step SARSA (Δ) markedly surpasses K-Step SARSA TD regarding average rewards. The favorable average reward of K-Step SARSA (Δ) demonstrates its superior suitability for policy optimization in the Blackjack-v1 environment, whereas the negative reward for K-Step SARSA TD signifies inferior performance.

In conclusion, the comparison of results across all four environments reveals a consistent advantage of K-Step SARSA (Δ) over K-Step SARSA TD in each case. This advantage is demonstrated by increased average rewards, quicker convergence, and enhanced stability in learning results. Environments defined by deterministic state transitions, such as CliffWalking-v0 and FrozenLake-v1, clearly showcase the advantages of K-Step SARSA (Δ). In contrast, more complex stochastic environments like Blackjack-v1 and Taxi-v3 further emphasize its robustness. The performance graph of deterministic environments, such as CliffWalking-v0 and FrozenLake-v1, resembles an exponential growth plateau curve due to predictable state transitions. On the contrary, the performance graph for sophisticated stochastic environments, such as Blackjack-v1 and Taxi-v3, resembles a flat line due to the uncertainty and unpredictability in state transitions.

Table 2 presents the average rewards along with confidence intervals (CI) for each algorithm within their respective environments. Kruskal statistical tests⁴⁵ were conducted at a significance level of $\alpha = 0.05$ to compare K-step SARSA TD and K-Step SARSA (Δ) baseline models.

Table 2. This table presents a comparative analysis of K-Step SARSA TD and K-Step SARSA (Δ) across multiple OpenAI Gym Toy Text environments.

Algorithm	Environments			
	Blackjack-v1	CliffWalking-v0	FrozenLake-v1	Taxi-v3
K-Step SARSA (Δ)	0.16 ± 0.00011	-1067.66 ± 5.041	0.081 ± 0.0017	-189.96 ± 1.39
K-Step SARSA TD	0.20 ± 0.000052	-1182.07 ± 8.75	0.063 ± 0.00079	-1754.70 ± 0.97

6.3 OpenAI Gym Atari Environments

To assess the efficacy of the proposed PPO-TD(λ , SARSA(Δ)) method, experiments were conducted in two Atari environments: *SeaquestNoFrameskip-v4* and *QbertNoFrameskip-v4*³, and the results were compared with those obtained from the standard PPO algorithm³⁹. The results shown in Figs. 7a and 7b show that in both environments, the proposed technique achieves better stability and better long-term rewards. The experiments tracked average rewards during training episodes ranging from 10,000 to 100,000. The experiments employ the code cited in Kostrikov⁴⁴.

In *SeaquestNoFrameskip-v4*, PPO-TD(λ , SARSA(Δ)) demonstrates superior performance compared to PPO-Standard at four out of five evaluation points, as illustrated in Fig. 7a. Following 100K episodes, the mean reward for PPO-TD(λ ,

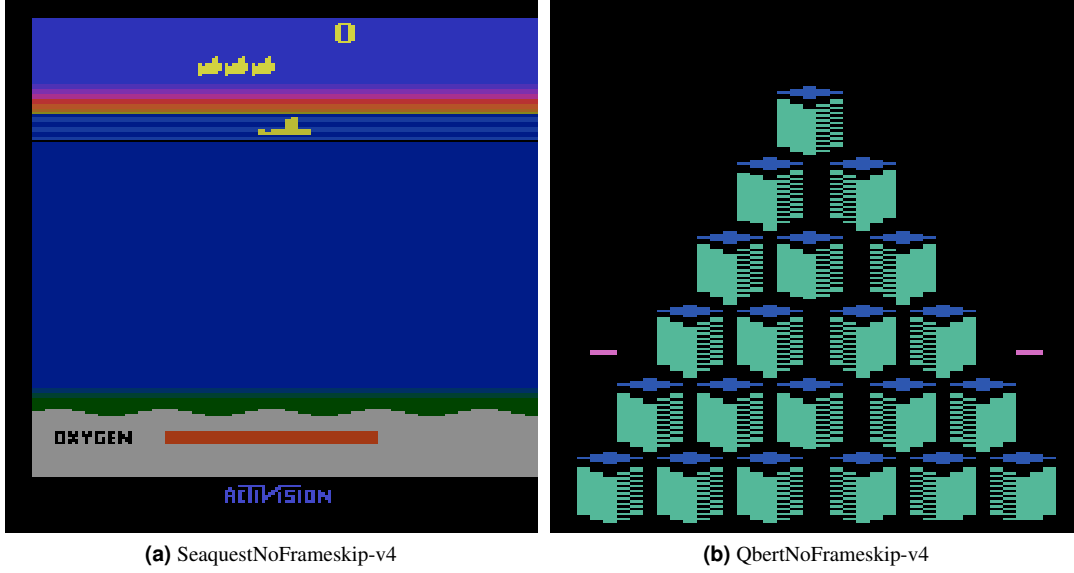


Figure 6. The OpenAI Gym Atari control framework is utilized for conducting long-horizon experiments. Arranged from left to right: (a) SeaquestNoFrameskip-v4 and (b) QbertNoFrameskip-v4.

SARSA(Δ) is around **14.35**, while PPO-Standard attains **14.25**, reflecting a **0.7% enhancement**. PPO-Standard has a pronounced peak after 60K episodes, achieving an average reward of around 15.25, after which it declines and plateaus. In contrast, PPO-TD(λ , SARSA(Δ)) exhibits more refined learning dynamics and sustains stable performance following its peak at 40K episodes.

The performance difference in *QbertNoFrameskip-v4* is more substantial, as illustrated in Fig. 7b. At 100K episodes, PPO-TD(λ , SARSA(Δ)) attains an average reward of approximately **275**, whereas PPO-Standard achieves about **210**, leading to an improvement of approximately **31%**. It is noteworthy that PPO-TD(λ , SARSA(Δ)) starts with lower rewards, around **155** at 10K episodes, in contrast to **210** for PPO-Standard. However, it exhibits a steady progress, finally surpassing the baseline after 60K episodes, and reaching a high of around **290** episodes after 80K episodes. The wider shaded region for the proposed technique in the earlier phases indicates more variance, which diminishes as training progresses—signifying improved convergence and learning efficiency with time.

In conclusion, these results highlight the advantages of integrating PPO and SARSA(Δ) with TD(λ , Δ). The suggested method enhances the stability of action-value estimates across various time scales and fosters superior long-term performance in dense-reward environments, especially in intricate games such as *Qbert*.

Table 3 shows the average rewards for each algorithm in their own environments, along with confidence intervals (CI). We used Kruskal statistical tests⁴⁵ at a significant level ($\alpha = 0.05$) to compare the PPO-TD(λ , SARSA(Δ))/PPO baseline models.

Table 3. Comparative analysis of PPO and PPO-TD(λ , SARSA(Δ)) across various OpenAI Gym Atari environments.

Algorithm	Atari Environments	
	Seaquest	Qbert
PPO-TD (λ , SARSA (Δ))	14.44 \pm 0.076	218.42 \pm 18.46
PPO	14.23 \pm 0.15	224.27 \pm 4.53

7 Discussion and Conclusions

By combining SARSA with TD(Δ) and various discount factors, SARSA(Δ) improves SARSA’s stability and performance. This method maintains the benefits of on-policy learning while enabling a more precise breakdown of action-values. To evaluate the effectiveness of SARSA(Δ), experiments were performed in both deterministic and stochastic environments from OpenAI Gym, as well as in a synthetic tabular environment—specifically, the five-star ring MDP. Furthermore, enhancements in performance within dense-reward environments of the Atari suite were demonstrated through a PPO-based variant of SARSA(Δ). The experimental results across all examined environments indicate that the proposed method, SARSA(Δ), surpasses k-step SARSA

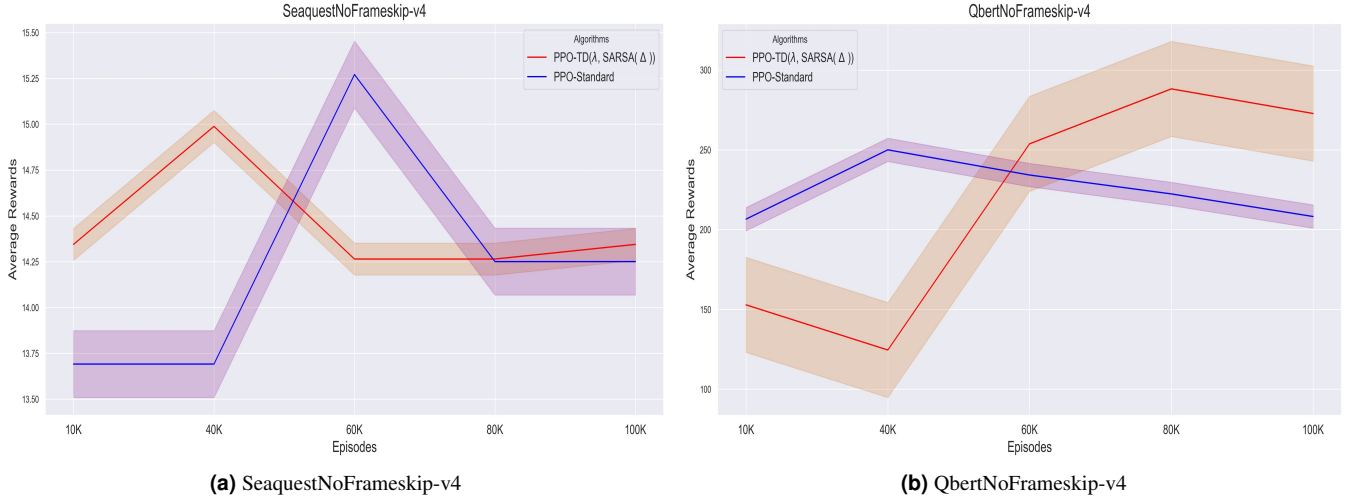


Figure 7. Comparison of performance between PPO-Standard and PPO-TD(λ , SARSA(Δ)) across two Atari games: (a) SeaquestNoFrameskip-v4 and (b) QbertNoFrameskip-v4. The x-axis denotes the quantity of training episodes (measured in thousands), whereas the y-axis illustrates the average rewards obtained. The shaded areas represent the standard deviation calculated from 10 runs with different random seeds.

TD by achieving higher average rewards, quicker convergence, and more consistent learning trajectories. The PPO-based variant of SARSA (Δ) also demonstrates superior performance compared to standard PPO algorithms in dense-reward Atari environments, as evidenced by the experimental results outlined in Section 6.3. This research contributes to our knowledge of multi-step RL algorithms and its applicability to diverse sequential decision-making tasks.

Limitations: The proposed SARSA(Δ) method has four limitations that require attention: (i) Memory Overhead: When using a large number of Δ components, the memory needs increase since the component-level updates scale linearly with the number of discount factors. This technique requires an extra $O(M)$ in storage compared to conventional SARSA. (ii) Stochastic Environments: Best results may not be attained through rigid scheduling of η_m in scenarios characterized by high unpredictability of rewards. In certain cases, adaptive scheduling of η_m can boost both the stability and the effectiveness of learning. (iii) Partial Observability Markov Decision Process (POMDP): Current formulations are inapplicable to environments characterized by partial observability, as they presuppose full observability of the Markov Decision Process (MDP). (iv) Extension to off-policy: There are numerous issues associated with adapting SARSA(Δ) to off-policy algorithms such as Q-learning. Off-policy learning has the potential to significantly increase variance in long-horizon components, which may subsequently lead to instability during the training process.

Future Directions: (i) Incorporation with Off-Policy Q-Learning: The adaptability of the SARSA(Δ) idea might be enhanced by applying it to off-policy algorithms, similar to the Q-learning version of TD(Δ). (ii) Applications to Multi-Agent Systems: In a multi-agent system, the agents coordinate on aligned subgoals by sharing D_m components. (iii) Adaptive Scheduling via Meta-Gradients: Performance in various environments may be improved by using meta-gradient approaches for dynamically adjusting η_m . (iv) Extension to Partially Observable: Adapting SARSA(Δ) for environments characterized by partial observability is an effective approach to enhance its relevance in real-world applications. We decided to take care of it later.

References

1. Romoff, J. *et al.* Separating value functions across time-scales. In *International Conference on Machine Learning*, 5468–5477 (PMLR, 2019).
2. Sutton, R. S., Barto, A. G. *et al.* *Reinforcement learning: An introduction*, vol. 2 (MIT press Cambridge, 2018).
3. Bellemare, M. *et al.* Unifying count-based exploration and intrinsic motivation. *Adv. neural information processing systems* **29** (2016).
4. Prokhorov, D. V. & Wunsch, D. C. Adaptive critic designs. *IEEE transactions on Neural Networks* **8**, 997–1007 (1997).
5. Mnih, V. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
6. Berner, C. *et al.* Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

7. François-Lavet, V., Fonteneau, R. & Ernst, D. How to discount deep reinforcement learning: Towards new dynamic strategies. *arXiv preprint arXiv:1512.02011* (2015).
8. Xu, Z., van Hasselt, H. P. & Silver, D. Meta-gradient reinforcement learning. *Adv. neural information processing systems* **31** (2018).
9. Van Seijen, H., Van Hasselt, H., Whiteson, S. & Wiering, M. A theoretical and empirical analysis of expected sarsa. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 177–184 (IEEE, 2009).
10. Fedus, W., Gelada, C., Bengio, Y., Bellemare, M. G. & Larochelle, H. Hyperbolic discounting and learning over multiple horizons. *arXiv preprint arXiv:1902.06865* (2019).
11. Kearns, M. J. & Singh, S. Bias-variance error bounds for temporal difference updates. In *COLT*, 142–147 (2000).
12. Sutton, R. S., Barto, A. G. *et al.* Introduction to reinforcement learning. vol. 135 (1998).
13. Sherstan, C., MacGlashan, J. & Pilarski, P. M. Generalizing value estimation over timescale. *Network* **2**, 3 (2018).
14. Ali, R. F. *et al.* Hyperbolic discounting in multi-agent reinforcement learning. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*.
15. Kim, M., Kim, J.-S., Choi, M.-S. & Park, J.-H. Adaptive discount factor for deep reinforcement learning in continuing tasks with uncertainty. *Sensors* **22**, 7266 (2022).
16. Amit, R., Meir, R. & Ciosek, K. Discount factor as a regularizer in reinforcement learning. In *International conference on machine learning*, 269–278 (PMLR, 2020).
17. Wang, J., Zhang, Q., Zhao, D., Zhao, M. & Hao, J. Dynamic horizon value estimation for model-based reinforcement learning. *arXiv preprint arXiv:2009.09593* (2020).
18. Dietterich, T. G. Hierarchical reinforcement learning with the maxq value function decomposition. *ArXiv* (1999).
19. Henderson, P. *et al.* Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. *ArXiv* (2017).
20. Hengst, B. Discovering hierarchy in reinforcement learning with hexq. In *International Conference on Machine Learning* (2002).
21. Reynolds, S. I. Decision boundary partitioning: Variable resolution model-free reinforcement learning. In *International Conference on Machine Learning* (1999).
22. Menache, I., Mannor, S. & Shimkin, N. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning* (2002).
23. Russell, S. J. & Zimdars, A. Q-decomposition for reinforcement learning agents. In *International Conference on Machine Learning* (2003).
24. van Seijen, H. *et al.* Hybrid reward architecture for reinforcement learning. *ArXiv* (2017).
25. Ali, R. F., Nafi, N. M., Duong, K. & Hsu, W. Efficient multi-horizon learning for off-policy reinforcement learning. In *Deep Reinforcement Learning Workshop NeurIPS 2022* (2022).
26. Ali, R. F., Duong, K., Nafi, N. M. & Hsu, W. Multi-horizon learning in procedurally-generated environments for off-policy reinforcement learning (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 16150–16151 (2023).
27. Seiler, K. M., Kong, F. H. & Fitch, R. Multi-horizon multi-agent planning using decentralised monte carlo tree search. *IEEE Robotics Autom. Lett.* (2024).
28. Benechehab, A., Paolo, G., Thomas, A., Filippone, M. & Kégl, B. Multi-timestep models for model-based reinforcement learning. *arXiv preprint arXiv:2310.05672* (2023).
29. Bonnet, C., Caron, P., Barrett, T., Davies, I. & Laterre, A. One step at a time: Pros and cons of multi-step meta-gradient reinforcement learning. *arXiv preprint arXiv:2111.00206* (2021).
30. Kobayashi, T. Adaptive and multiple time-scale eligibility traces for online deep reinforcement learning. *Robotics Auton. Syst.* **151**, 104019 (2022).
31. Sutton, R. S. Learning to predict by the methods of temporal differences. *Mach. learning* **3**, 9–44 (1988).
32. Tsitsiklis, J. & Van Roy, B. Analysis of temporal-difference learning with function approximation. *Adv. neural information processing systems* **9** (1996).

33. Bellman, R. A markovian decision process. *J. mathematics mechanics* 679–684 (1957).
34. Sutton, R. S. *Temporal credit assignment in reinforcement learning* (University of Massachusetts Amherst, 1984).
35. Schulman, J., Moritz, P., Levine, S., Jordan, M. I. & Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *CoRR abs/1506.02438* (2015).
36. Sutton, R. S., McAllester, D. A., Singh, S. & Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems* (1999).
37. Konda, V. R. & Tsitsiklis, J. N. Actor-critic algorithms. In *Neural Information Processing Systems* (1999).
38. Mnih, V. *et al.* Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning* (2016).
39. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
40. Henderson, P., Romoff, J. & Pineau, J. Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *arXiv preprint arXiv:1810.02525* (2018).
41. Kearns, M. & Singh, S. Near-optimal reinforcement learning in polynomial time. *Mach. learning* **49**, 209–232 (2002).
42. Brockman, G. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).
43. Peter Henderson, W.-D. C. Value and policy iterations. <https://github.com/Breakend/ValuePolicyIterationVariations> (2024).
44. Kostrikov, I. Pytorch implementations of reinforcement learning algorithms. <https://github.com/ikostrikov/pytorch-a2c-ppo-acktr> (2024).
45. Kruskal, W. H. & Wallis, W. A. Use of ranks in one-criterion variance analysis. *J. Am. statistical Assoc.* **47**, 583–621 (1952).

Acknowledgments

This research is funded by the Innovation Teams of Ordinary Universities in Guangdong Province under Grants (2021KCXTD038, 2023KCXTD022), the Key Laboratory of Ordinary Universities in Guangdong Province (2022KSYS003), Key Discipline Research Ability Improvement Project of Guangdong Province (2021ZDJS043, 2022ZDJS068), the Natural Science Foundation of Guangdong Province (2022A1515010990), the Research Fund of the Department of Education of Guangdong Province Grants (2022KTSCX079, 2023ZDZX1013, 2022ZDZX3012, 2022ZDZX3011, 2023ZDZX2038), Chaozhou Engineering Technology Research Center (z25025) and Hanshan Normal University project (XY202105). We express our gratitude to the editors and referees for their invaluable suggestions and remarks.

Author contributions statement

M.H. participated in all stages, from Conceptualization design to writing—original draft preparation. G.Z., and X.D. funding acquisition. X.C. and Y.Z. supervision and project administration. W.H., L.M., S.Q., Z.Z., and P.W. investigation. Z.U. and N.J. validation and formal analysis.

Data availability

The corresponding authors may be contacted to request data related to this paper.

Competing interests

The authors disclose no conflicting interests.

Appendix

A PROOFS

Proof: Theorem 1

The proof utilizes induction. Initially, it is necessary to establish that the base case is valid at $n=0$. This statement is evidently valid given our initial assumption, particularly in the context of zero initialization. At a specific time-step n , we assume the validity of the statement, namely, $\sum_{m=0}^M \theta_n^m = \theta_n^\eta$. We will now demonstrate that it also holds at the subsequent time step, $n+1$.

$$\begin{aligned} \sum_{m=0}^M \theta_{n+1}^m &= \sum_{m=0}^M \left(\theta_n^m + \alpha_m \left(G_n^{M, \lambda_m} - \hat{D}_m(s_n, a_n) \right) \phi(s_n, a_n) \right) \\ D_m(s, a) &= Q_{\eta_m}(s, a) - Q_{\eta_{m-1}}(s, a) \text{ and using Eq. 21} \\ &= \theta_n^\eta + \sum_{m=0}^M \alpha_m \left(\sum_{k=n}^{\infty} (\lambda_m \eta_m)^{k-n} \delta_k^m \right) \phi(s_n, a_n), \text{ Assumption based on induction.} \\ &= \theta_n^\eta + \alpha \sum_{k=n}^{\infty} (\lambda \eta)^{k-n} \underbrace{\left(\sum_{m=0}^M \delta_k^m \right)}_{*} \phi(s_n, a_n), \text{ Because of } \alpha_m = \alpha, \lambda_m \eta_m = \lambda \eta, \forall m \end{aligned}$$

To show that $\sum_{m=0}^M \theta_{n+1}^m = \theta_{n+1}^\eta$, we must establish that the term $(*) = \sum_{m=0}^M \delta_k^m$ is equivalent to the conventional TD error δ_k^η .

$$\begin{aligned} \sum_{m=0}^M \delta_k^m &= r_k + \eta_0 \hat{Q}_{\eta_0}(s_{k+1}, a_{k+1}) - \hat{Q}_{\eta_0}(s_k, a_k) + \sum_{m=1}^M \left((\eta_m - \eta_{m-1}) \sum_{q=0}^{m-1} \langle \theta_n^q, \phi(s_{k+1}, a_{k+1}) \rangle \right. \\ &\quad \left. + \eta_m \langle \theta_n^m, \phi(s_{k+1}, a_{k+1}) \rangle - \langle \theta_n^m, \phi(s_k, a_k) \rangle \right) \\ &= r_k + \eta_0 \hat{Q}_{\eta_0}(s_{k+1}, a_{k+1}) + \left\langle \sum_{m=1}^M \left(\eta_m \sum_{q=0}^m \theta_n^q - \eta_{m-1} \sum_{q=0}^{m-1} \theta_n^q \right), \phi(s_{k+1}, a_{k+1}) \right\rangle \\ &\quad - \left\langle \sum_{m=0}^M \theta_n^m, \phi(s_k, a_k) \right\rangle \\ &= r_k + \eta_0 \hat{Q}_{\eta_0}(s_{k+1}, a_{k+1}) + \eta_m \left\langle \sum_{m=0}^M \theta_n^m, \phi(s_{k+1}, a_{k+1}) \right\rangle - \eta_0 \left\langle \theta_n^0, \phi(s_{k+1}, a_{k+1}) \right\rangle \\ &\quad - Q_\eta(s_k, a_k) \\ &= r_k + \eta_0 \hat{Q}_{\eta_0}(s_{k+1}, a_{k+1}) + \eta \hat{Q}_\eta(s_{k+1}, a_{k+1}) - \eta_0 \hat{Q}_{\eta_0}(s_{k+1}, a_{k+1}) - \hat{Q}_\eta(s_k, a_k) \\ &= r_k + \eta \hat{Q}_\eta(s_{k+1}, a_{k+1}) - \hat{Q}_\eta(s_k, a_k) \\ &= \delta_k^\eta \end{aligned}$$

□

Proof: Theorem 2

The following is the definition of the Bellman operator:

$$T = r + \eta P$$

The discount factor, the expected reward function, and the transition probability operator that is induced by the policy π are denoted by the symbols η , r , and P , respectively. A geometric weighted sum of T is the definition of the $TD(\lambda)$ operator, which can be expressed as follows:

$$T = (1 - \lambda) \sum_{k=0}^{\infty} \lambda^k (T)^{k+1} \quad (30)$$

It may be inferred that $\lambda \in [0, 1]$ is necessary for the aforementioned sum to be finite. A corresponding definition of T_λ is as follows for every function D :

$$T_\lambda D = D + (I - \lambda \eta P)^{-1} (TD - D) \quad (31)$$

The separated action-value function is denoted by D in this form, and the update is given in terms of the operator T_λ , which is responsible for applying time-scale separation. In the event that $0 \leq \lambda \eta < 1$, the formula is considered to be well-defined. This is because the spectral norm of the operator $\lambda \eta P$ is less than 1, and thus, Eq. $I - \lambda \eta P$ is invertible. On the other hand, when the value of λ is bigger than one, the equivalence between Eqs. 30 and 31 is lost. This is not a problem because, in fact, training is based on the TD error, which, according to expectations, corresponds to the definition of T_λ that is provided in Eq. 31. Now, let's have a look at the contraction property that the operator T_λ possesses. To begin, the Eq. designated as 31 can be rewritten as follows:

$$T_\lambda = (I - \lambda \eta P)^{-1}(TD - \lambda \eta PD) \quad (32)$$

The contraction property is demonstrated by examining two action-value functions, D_1 and D_2 . The distinction between the updates of $T_\lambda D_1$ and $T_\lambda D_2$ is:

$$\begin{aligned} T_\lambda D_1 - T_\lambda D_2 &= (I - \lambda \eta P)^{-1} \left(TD_1 - TD_2 - \lambda \eta P(D_1 - D_2) \right) \\ &= (I - \lambda \eta P)^{-1} \left(\eta P(D_1 - D_2) - \lambda \eta P(D_1 - D_2) \right) \\ &= (I - \lambda \eta P)^{-1} \left(\eta(1 - \lambda)P(D_1 - D_2) \right) \end{aligned}$$

The following is derived:

$$\|T_\lambda D_1 - T_\lambda D_2\| \leq \frac{\eta|1 - \lambda|}{1 - \lambda \eta} \|D_1 - D_2\| \text{ where } P = I$$

It is established that $0 \leq \lambda \leq 0$ constitutes a contraction. For $\lambda > 1$, the following condition must be satisfied:

$$\begin{aligned} \frac{\eta(\lambda - 1)}{1 - \lambda \eta} < 1 &\Rightarrow \eta\lambda - \eta < 1 - \lambda\eta \\ &\Rightarrow 2\lambda\eta < 1 + \eta \Rightarrow \lambda < \frac{1 + \eta}{2\eta} \end{aligned} \quad (33)$$

Consequently, T_λ constitutes a contraction if $0 \leq \lambda < \frac{1 + \eta}{2\eta}$. This directly indicates that for $\eta < 1$, $\eta\lambda < 1$.

□

Proof: Theorem 3

In SARSA, the Q – value function, $Q(s, a)$, is estimated using samples of state-action pairs and their corresponding rewards. Let $Q_n(s, a)$ denote the estimate of the Q – value at time n . For n samples, by Hoeffding's inequality provides a guarantee for a variable that is bounded between $[-1, +1]$, that, the probability that the sample mean deviates from the true mean by more than ϵ is expressed as follows:

$$Q^\pi(s, a) = \mathbf{E} \left[\eta_0 + \eta r_1 + \dots + \eta^{k-1} r_{k-1} + \eta^k Q^\pi(s_k, a_k) \right]$$

Here, the expectations are over a random trajectory under π ; thus $\mathbf{E}[r_t] (t \leq k-1)$ denotes the expected value of the t th reward received, while $\mathbf{E}[Q^\pi(s_k, a_k)]$ is the expected value of the true value function at the k th state-action reached. The phased TD(k) update sums the terms $\eta^t \left(\frac{1}{n} \sum_{i=1}^n r_i^t \right)$, whose expectations are exactly the $\eta^t \mathbf{E}[r_t]$ appearing above¹¹.

$$\begin{aligned} \mathbb{P} \left(\left| \frac{1}{n} \sum_{i=1}^n r_i^{(x)} - \mathbb{E}[r_i] \right| \geq \epsilon \right) &\leq 2e^{\left(-\frac{2n^2 \epsilon^2}{\sum_{i=1}^n (b-a)^2} \right)} \text{ where } a=-1 \text{ and } b=+1 \\ \mathbb{P} \left(\left| \frac{1}{n} \sum_{i=1}^n r_i^{(x)} - \mathbb{E}[r_i] \right| \geq \epsilon \right) &\leq 2e^{\left(-\frac{2n^2 \epsilon^2}{n^2} \right)} \end{aligned} \quad (34)$$

Assuming that n and the probability of surpassing an ϵ value is fixed to be atmost δ , we can derive the corresponding value of ϵ :

$$2e^{\left(-\frac{2n^2\epsilon^2}{n2^2}\right)} = \delta$$

$$e^{\left(-\frac{2n^2\epsilon^2}{n2^2}\right)} = \frac{\delta}{2}$$

Taking the natural logarithm

$$\ln\left(e^{\left(-\frac{2n^2\epsilon^2}{n2^2}\right)}\right) = \ln\left(\frac{\delta}{2}\right)$$

$$-\frac{2n^2\epsilon^2}{n2^2} = \ln\frac{\delta}{2}$$

$$-\frac{n\epsilon^2}{2} = \ln\frac{\delta}{2}$$

$$-\frac{n\epsilon^2}{2} = \ln\frac{\delta}{2}$$

$$\frac{n\epsilon^2}{2} = -\ln\frac{\delta}{2}$$

$$\frac{n\epsilon^2}{2} = -\left(\ln(\delta) - \ln(2)\right)$$

$$\frac{n\epsilon^2}{2} = -\left(\ln(\delta) + \ln(2)\right)$$

$$\frac{n\epsilon^2}{2} = \left(\ln(2) - \ln(\delta)\right)$$

$$\frac{n\epsilon^2}{2} = \ln\frac{2}{\delta}$$

$$n\epsilon^2 = 2\log\frac{2}{\delta}$$

$$\epsilon^2 = \frac{2\log\frac{2}{\delta}}{n}$$

$$\epsilon = \sqrt{\frac{2\log\frac{2}{\delta}}{n}} \quad (35)$$

So by Hoeffding's inequality, for n samples, the following holds with probability at least $1 - \delta$,

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{x=1}^n r_i^{(x)} - \mathbb{E}[r_i]\right|\right) \leq \epsilon = \sqrt{\frac{2\log\frac{2}{\delta}}{n}} \quad (36)$$

Since the analysis involves k different state-action pairs, a union bound is applied. To ensure the probability holds for all k hypotheses, adjust δ to δ/k :

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{x=1}^n r_i^{(x)} - \mathbb{E}[r_i]\right|\right) \leq \epsilon = \sqrt{\frac{2\log\frac{2k}{\delta}}{n}}$$

It is now assumed that all $\mathbb{E}[r_i]$ terms are estimated to at least ϵ accuracy. Substituting this back into the definition of the k -step TD update we get

$$\hat{Q}_{n+1}(s, a) - Q(s, a) = \frac{1}{n} \sum_{x=1}^n \left(r_0 + \eta r_1 + \dots + \eta^{k-1} r_{k-1} + \eta^k Q_n(s_k, a_k) \right) - Q(s, a)$$

$$= \sum_{i=0}^{k-1} \eta^i \left(\frac{1}{n} \sum_{i=1}^n r_i^{(x)} - \mathbb{E}[r_i] \right) + \eta^k \left(\frac{1}{n} \sum_{i=1}^n Q_n(s_k^i, a_k^i) - \mathbb{E}[Q(s_k, a_k)] \right) \quad (37)$$

where in the second line we re-expressed the value in terms of a sum of k rewards. We now upper bounded the difference from $E[r_i]$ by ε to get

$$\hat{Q}_{n+1}(s, a) - Q(s, a) \leq \sum_{l=0}^{k-1} \eta^l \varepsilon + \eta^k \left(\frac{1}{n} \sum_{i=1}^n Q_n(s_k^i, a_k^i) - \mathbb{E}[Q(s_k, a_k)] \right)$$

The variance term arises from the deviation of the empirical average of rewards from the true expected reward:

$$\begin{aligned} & \varepsilon \left(\frac{1 - \eta^k}{1 - \eta} \right) \\ & \leq \varepsilon \left(\frac{1 - \eta^k}{1 - \eta} \right) + \eta^k \left(\frac{1}{n} \sum_{i=1}^n Q_n(s_k^i, a_k^i) - \mathbb{E}[Q(s_k, a_k)] \right) \end{aligned} \quad (38)$$

The bias term is propagated through bootstrapping:

$$\eta^k \Delta_{n-1}^{\hat{Q}_\eta}$$

and then the second term is bounded by $\Delta_{n-1}^{\hat{Q}_\eta}$ by assumption. Therefore, the combination of these terms results in the comprehensive bound for SARSA:

$$\Delta_n^{\hat{Q}_\eta} \leq \varepsilon \left(\frac{1 - \eta^k}{1 - \eta} \right) + \eta^k \Delta_{n-1}^{\hat{Q}_\eta} \quad (39)$$

□

Proof: Theorem 4

Phased $TD(\Delta)$ update rules for $m \geq 1$:

$$\hat{D}_{m,n}(s, a) = \frac{1}{n} \sum_{x=1}^n \left(\sum_{i=1}^{k_m-1} (\eta_m^i - \eta_{m-1}^i) r_i^{(x)} + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) \hat{Q}_{\eta_{m-1}}(s_k^{(x)}, a_k^{(x)}) + \eta_m^{k_m} \hat{D}_m(s_k^{(x)}, a_k^{(x)}) \right) \quad (40)$$

According to the multi-step update rule 16 for $m \geq 1$:

$$D_m(s_n, a_n) = \mathbb{E} \left[\sum_{i=1}^{k_m-1} (\eta_m^i - \eta_{m-1}^i) r_i + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) Q_{\eta_{m-1}}(s_k, a_k) + \eta_m^{k_m} D_m(s_k, a_k) \right] \quad (41)$$

Then, subtracting the two expressions gives for $m \geq 1$:

$$\begin{aligned} \hat{D}_{m,n}(s, a) - D_m(s_n, a_n) &= \sum_{i=1}^{k_m-1} (\eta_m^i - \eta_{m-1}^i) \left(\frac{1}{n} \sum_{x=1}^n r_i^{(x)} - \mathbb{E}[r_i] \right) + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) \\ & \quad \left(\sum_{q=0}^{m-1} \hat{D}_q(s_k^{(x)}, a_k^{(x)}) - \mathbb{E}[D_m(s_k, a_k)] \right) + \eta_m^{k_m} \left(D_m(s_k^{(x)}, a_k^{(x)}) - \mathbb{E}[D_m(s_k, a_k)] \right) \end{aligned}$$

Assume that $k_0 \leq k_1 \leq \dots k_m = k$, the D estimates share at most $k_m = k$ reward terms $\frac{1}{n} \sum_{x=1}^n r_i^{(x)}$, Using Hoeffding inequality and union bound, it follows that, with probability $1 - \delta$, each k empirical average reward $\frac{1}{n} \sum_{x=1}^n r_i^{(x)}$ deviates from the true expected reward $\mathbb{E}[r_i]$ by at most $\varepsilon = \sqrt{\frac{2 \log(2k/\delta)}{n}}$. Hence, with probability $1 - \delta$, $\forall m \geq 1$, the following holds:

$$\begin{aligned} \Delta_n^m &\leq \varepsilon \sum_{i=1}^{k_m-1} (\eta_m^i - \eta_{m-1}^i) + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) \sum_{q=0}^{m-1} \Delta_{n-1}^q + \eta_m^{k_m} \Delta_{n-1}^m \\ &= \varepsilon \left(\frac{1 - \eta_m^{k_m}}{1 - \eta_m} - \frac{1 - \eta_{m-1}^{k_m}}{1 - \eta_{m-1}} \right) + (\eta_m^{k_m} - \eta_{m-1}^{k_m}) \sum_{q=0}^{m-1} \Delta_{n-1}^q + \eta_m^{k_m} \Delta_{n-1}^m \end{aligned} \quad (42)$$

$$\text{and } \Delta_n^0 \leq \varepsilon \frac{1 - \eta_0^{k_0}}{1 - \eta_0} + \eta_0^{k_0} \Delta_{n-1}^0$$

Summing the two previous inequalities gives:

$$\sum_{m=0}^M \Delta_n^m \leq \varepsilon \frac{1 - \eta_0^{k_0}}{1 - \eta_0} + \varepsilon \sum_{m=1}^M \left(\frac{1 - \eta_m^{k_m}}{1 - \eta_m} - \frac{1 - \eta_{m-1}^{k_m}}{1 - \eta_{m-1}} \right) + \sum_{m=1}^M (\eta_m^{k_m} - \eta_{m-1}^{k_m}) \sum_{q=0}^{m-1} \Delta_{n-1}^q + \eta_m^{k_m} \Delta_{n-1}^m$$

$$= \underbrace{\varepsilon \frac{1 - \eta_M^{k_M}}{1 - \eta_M} + \varepsilon \sum_{m=0}^{M-1} \frac{\eta_m^{k_{m+1}} - \eta_m^{k_m}}{1 - \eta_m}}_{(*) \text{variance term}} + \underbrace{\sum_{m=1}^M \left(\eta_m^{k_m} - \eta_{m-1}^{k_m} \right) \sum_{q=0}^{m-1} \Delta_{n-1}^q + \eta_m^{k_m} \Delta_{n-1}^m}_{(**) \text{bias term}} \quad (43)$$

Let's focus now further on the bias term (**)

$$\begin{aligned} & \sum_{m=1}^M \left(\eta_m^{k_m} - \eta_{m-1}^{k_m} \right) \sum_{q=0}^{m-1} \Delta_{n-1}^q + \eta_m^{k_m} \Delta_{n-1}^m = \sum_{q=0}^{M-1} \sum_{m=q+1}^M \left(\eta_m^{k_m} - \eta_{m-1}^{k_m} \right) \Delta_{n-1}^q + \sum_{m=1}^M \eta_m^{k_m} \Delta_{n-1}^m \\ &= \sum_{q=0}^{M-1} \Delta_{n-1}^q \left(\sum_{m=q+1}^M \eta_m^{k_m} - \sum_{m=q}^{M-1} \eta_m^{k_{m+1}} \right) + \sum_{m=1}^M \eta_m^{k_m} \Delta_{n-1}^m \\ &= \sum_{q=0}^{M-1} \Delta_{n-1}^q \left(\sum_{m=q+1}^{M-1} (\eta_m^{k_m} - \eta_m^{k_{m+1}}) + (\eta_M^{k_M} - \eta_q^{k_{q+1}}) \right) + \sum_{m=1}^M \eta_m^{k_m} \Delta_{n-1}^m \\ &= \sum_{q=0}^{M-1} \sum_{m=q+1}^{M-1} (\eta_m^{k_m} - \eta_m^{k_{m+1}}) \Delta_{n-1}^q + \eta_M^{k_M} \sum_{m=0}^M \Delta_{n-1}^m + \sum_{m=0}^{M-1} (\eta_m^{k_m} - \eta_m^{k_{m+1}}) \Delta_{n-1}^m \\ &= \sum_{q=0}^{M-1} \sum_{m=q}^{M-1} (\eta_m^{k_m} - \eta_m^{k_{m+1}}) \Delta_{n-1}^q + \eta_M^{k_M} \sum_{m=0}^M \Delta_{n-1}^m \\ &= \sum_{m=0}^{M-1} (\eta_m^{k_m} - \eta_m^{k_{m+1}}) \sum_{q=0}^m \Delta_{n-1}^q + \eta_M^{k_M} \sum_{m=0}^M \Delta_{n-1}^m \end{aligned} \quad (44)$$

Finally, the following is obtained:

$$\sum_{m=0}^M \Delta_n^m \leq \underbrace{\varepsilon \left(\frac{1 - \eta^k}{1 - \eta} \right) + \varepsilon \left(\sum_{m=0}^{M-1} \frac{\eta_m^{k_{m+1}} - \eta_m^{k_m}}{1 - \eta_m} \right)}_{\text{variance reduction}} + \underbrace{\sum_{m=0}^{M-1} \left(\eta_m^{k_m} - \eta_m^{k_{m+1}} \right) \sum_{q=0}^m \Delta_{n-1}^q + \eta^k \sum_{m=0}^M \Delta_{n-1}^m}_{\text{bias introduction}} \quad (45)$$

□

B Hyperparameters

The experiments primarily utilize the hyperparameters recommended by Romoff 771 et al.¹, Schulman et al.³⁹ and Kostrikov⁴⁴. The experiments utilize the following randomly generated seeds: 125125, 513, 90135, 81212, 3523401, 15709, 17, 0, 8412, and 1153780. Additional hyperparameter configurations are detailed in Table 4.

Table 4. Hyperparameter configurations and their corresponding values.

Hyperparameter	Value
seed (10)	np.random.randint(40, 45)
Learning Rate (α)	[0.1, 0.3, 0.5, 0.7, 1.0]
Discount Factor (η)	[0.992, 0.95, 0.996, 0.99, 1.0]
k-Step	[1, 4, 8, 16, 32, 128, 256]
Number of Episodes	100000
Max Epsilon	0.4
Clipping parameter	0.1
Number of actors	8
Horizon(T)	129
Number of Epochs	3
Lambda (λ)	0.95