Deep Reinforcement Learning with Gradient Eligibility Traces

Anonymous authors
Paper under double-blind review

Keywords: Deep RL, Gradient TD, Eligibility Traces, PPO

Summary

Achieving fast and stable off-policy learning in deep reinforcement learning (RL) is challenging. Most existing methods rely on semi-gradient temporal-difference (TD) methods for their simplicity and efficiency but are consequently susceptible to divergence. While more principled approaches like Gradient TD (GTD) methods have strong convergence guarantees, they have rarely been used in deep RL. Recent work introduced the Generalized Projected Bellman Error ($\overline{\text{GPBE}}$), enabling GTD methods to work efficiently with nonlinear function approximation. However, this work is only limited to one-step methods, which are slow at credit assignment and require a large number of samples. In this paper, we extend the $\overline{\text{GPBE}}$ objective to use multistep credit assignment based on the λ-return and derive three gradient-based methods that optimize this new objective. We provide both a forward-view formulation compatible with experience replay and a backward-view formulation compatible with streaming algorithms. Finally, we evaluate the proposed algorithms and show that they outperform both PPO and StreamQ in Mujoco and MinAtar environments, respectively.

Contribution(s)

1. We extend the $\overline{\text{GPBE}}$ to incorporate multistep credit assignment based on λ-returns, defining a new objective, the $\overline{\text{GPBE}}(\lambda)$ (Section 3).
   Context: Patterson et al. (2022b) introduced the $\overline{\text{GPBE}}$, which unifies and generalizes previously known objectives for value estimation. However, it was only defined for the 1-step TD error.
2. We derive three Gradient TD algorithms that optimize our proposed objective. We derive both the forward view with the λ-return (Section 4) and the backward view with eligibility traces (Section 6).
   Context: Gradient TD methods were originally introduced with linear function approximation (Sutton et al., 2009), with a limited extension to nonlinear function approximation that required second-order information (Maei et al., 2009). The recent work by Patterson et al. (2022b) extended these methods to non-linear function approximation without a need for second-order information. However, it was limited to the 1-step TD error.
3. We introduce Gradient PPO, a policy gradient algorithm that uses our sound forward-view value estimation algorithms (Section 5).
   Context: PPO (Schulman et al., 2017) is a widely-used policy gradient method that relies on semi-gradient TD updates for value estimation. We build on PPO by replacing the value estimation component with a new one that uses Gradient TD methods. This change required non-trivial modification to PPO, resulting in our new algorithm, Gradient PPO. Gradient PPO is the first policy gradient method that uses Gradient TD algorithms in a deep RL setting with a replay buffer.
4. We introduce QRC(λ), which uses our backward-view eligibility traces and is suitable for streaming settings (Section 6).
   Context: Backward-view algorithms can make updates on each time step without delay, making them efficient in streaming settings (Elsayed et al., 2024). QRC(λ) is the first backward-view algorithm that uses Gradient TD methods in the streaming deep RL setting.

Deep Reinforcement Learning with Gradient Eligibility Traces

Anonymous authors
Paper under double-blind review

Abstract

Achieving fast and stable off-policy learning in deep reinforcement learning (RL) is challenging. Most existing methods rely on semi-gradient temporal-difference (TD) methods for their simplicity and efficiency but are consequently susceptible to divergence. While more principled approaches like Gradient TD (GTD) methods have strong convergence guarantees, they have rarely been used in deep RL. Recent work introduced the Generalized Projected Bellman Error ($\overline{\text{GPBE}}$), enabling GTD methods to work efficiently with nonlinear function approximation. However, this work is only limited to one-step methods, which are slow at credit assignment and require a large number of samples. In this paper, we extend the $\overline{\text{GPBE}}$ objective to use multistep credit assignment based on the λ-return and derive three gradient-based methods that optimize this new objective. We provide both a forward-view formulation compatible with experience replay and a backward-view formulation compatible with streaming algorithms. Finally, we evaluate the proposed algorithms and show that they outperform both PPO and StreamQ in Mujoco and MinAtar environments, respectively.

1 Introduction

Estimating the value function is a fundamental component of most RL algorithms. All value-based methods depend on estimating the action-value function for some target policy and then acting greedily with respect to those estimated values. Even in policy gradient methods, where a parameterized policy is learned, most algorithms learn a value function along with the policy. Many RL algorithms use semi-gradient temporal difference (TD) learning algorithms for value estimation, despite known divergence issues under nonlinear function approximation (Tsitsiklis & Van Roy, 1996) and under off-policy sampling (Baird, 1995), both of which frequently arise in modern deep RL settings.

There have been significant advances towards deriving TD algorithms that are sound. This progress occurred once it became clear what objective underlies the TD solution. For a brief history, the mean squared Bellman error ($\overline{\text{BE}}$) was an early objective, that produces a different solution than the TD fixed point but similarly aims to satisfy the Bellman equation. However, the $\overline{\text{BE}}$ was not widely-used because it is difficult to optimize without a simulator due to the double-sampling problem (Baird, 1995). The mean squared *projected* Bellman error ($\overline{\text{PBE}}$) for linear function approximation was introduced later, and a class of Gradient TD methods were derived to optimize this objective (Sutton et al., 2009). An early attempt to extend Gradient TD methods to nonlinear function approximation required computing Hessian-vector products (Maei et al., 2009). Patterson et al. (2022b) then introduced the generalized $\overline{\text{PBE}}$ ($\overline{\text{GPBE}}$), which is based on the conjugate form of the $\overline{\text{BE}}$ (Dai et al., 2017), making it much simpler to derive Gradient TD methods for the nonlinear setting. This generalized objective was further extended to allow for robust losses in the Bellman error (Patterson et al., 2022a) and is a promising avenue for the development of sound value-estimation algorithms. The $\overline{\text{GPBE}}$ and robust extensions, however, have only been explored for the one-step setting.

Table 1: Related Gradient TD literature. Our paper is the first to define and optimize the GPBE(λ) objective for nonlinear function approximation (see Section 4).

	Linear Function Approximation		Nonlinear Function Approximation	
Objective	**1-step**	**λ-return**	**1-step**	**λ-return**
PBE	(Sutton et al., 2009)	(Maei & Sutton, 2010)	(Maei et al., 2009)	Our paper
GPBE	(Patterson et al., 2022b)	Our paper	(Patterson et al., 2022b)	Our paper

In this paper, we extend the $\overline{\text{GPBE}}$ to incorporate multistep credit assignment using λ-returns. Table 1 summarizes the algorithmic gaps that we fill. We derive similar gradient variants as were derived for the one-step $\overline{\text{GPBE}}$ (Patterson et al., 2022b), but now also need to consider forward-view and backward-view updates for our proposed objective, $\overline{\text{GPBE}}(\lambda)$. We introduce Gradient PPO, a policy gradient algorithm that modifies PPO to use our sound forward-view value estimation algorithms. We also introduce QRC(λ), which uses backward-view eligibility traces and is suitable for streaming settings. We show that Gradient PPO significantly outperforms PPO in two Mujoco environments and is comparable in two others. We show that QRC(λ) is significantly better in several MinAtar environments than StreamQ (Elsayed et al., 2024), a recent algorithm combining Q(λ) with a new optimizer and an initialization scheme for better performance in streaming settings. We investigate multiple variants of our forward-view and backward-view algorithms, and as was concluded for $\overline{\text{GPBE}}(0)$ (Ghiassian et al., 2020; Patterson et al., 2022b), find that a variant based on regularized corrections called TDRC consistently outperforms the other variants. This work provides a clear conclusion on how to incorporate gradient TD methods with eligibility traces into deep RL methods and offers two new promising algorithms.

2 Background

We consider the Markov Decision Process (MDP) formalism where the agent-environment interactions are described by the tuple $(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$. At each time step, $t = 1, 2, 3, \ldots$, the agent observes a state, $S_t \in \mathcal{S}$, and takes an action, $A_t \in \mathcal{A}$ according to a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $\mathcal{S}$ and $\mathcal{A}$ are finite sets of states and actions, respectively. Based on S_t and A_t, the environment transitions to a new state, $S_{t+1} \in \mathcal{S}$, and yields a reward, $R_{t+1} \in \mathcal{R}$, with probability $p(S_{t+1}, R_{t+1} \mid S_t, A_t)$. The value of a policy is defined as $v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s], \forall s \in \mathcal{S}$, where the return, $G_t \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} \gamma^i R_{t+1+i}$, is the discounted sum of future rewards from time t and γ is a discount factor, $\gamma \in [0, 1]$.

The agent typically estimates the value function using a differentiable parameterized function, such as a neural network. We define the parameterized value function as $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$, where $\boldsymbol{w} \in \mathbb{R}^{d_{\boldsymbol{w}}}$ is a weight vector and $d_{\boldsymbol{w}} < |\mathcal{S}|$. One objective to learn this value function is the mean squared Bellman error ($\overline{\text{BE}}$)

$$\overline{\text{BE}}(\boldsymbol{w}) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d(s)\, \mathbb{E}_\pi[\delta \mid S = s]^2\,, \tag{1}$$

where d is the state distribution[1] and δ is the TD error for a transition (S, A, S', R). The δ can be different depending on the algorithm. For state-value prediction, we use $\delta \stackrel{\text{def}}{=} R + \gamma \hat{v}(S', \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w})$. For control, to learn optimal action-values $q_*(s, a)$, we use $\delta \stackrel{\text{def}}{=} R + \gamma \max_{a' \in \mathcal{A}} \hat{q}(S', a', \boldsymbol{w})) - \hat{q}(S, A, \boldsymbol{w})$. For control, we would additionally condition on $A = a$ above and sum over (s, a) instead of s, but for simplicity of exposition, we only show the objectives for $\hat{v}$. We can not generally reach zero $\overline{\text{BE}}$, unless the true value function is representable by our parameterized function class. The $\overline{\text{BE}}$ objective is difficult to optimize, due to the double sampling issue, and we instead consider a more practical objective called the $\overline{\text{GPBE}}$.

[1]Note that we write the expectation with a sum to make the notation more accessible, but this can be generalized to continuous state spaces using integrals.

The $\overline{\text{GPBE}}$ objective generalizes and unifies several objectives and extends Gradient TD methods to nonlinear function approximation (Patterson et al., 2022b). The $\overline{\text{GPBE}}$ builds on the work by Dai et al. (2017) that avoids the double sampling by reformulating the $\overline{\text{BE}}$ using its conjugate form with an auxiliary variable h. Using the fact that the biconjugate of a quadratic function is $x^2 = \max_{h \in \mathbb{R}} 2xh - h^2$, we can re-express the $\overline{\text{BE}}$ as

$$\overline{\text{BE}}(\boldsymbol{w}) \stackrel{\text{def}}{=} \max_{h \in \mathcal{F}_{\text{all}}} \sum_{s \in S} d(s) \Big(2\, \delta_\pi(s)\, h(s) - h(s)^2 \Big) , \tag{2}$$

where $\mathcal{F}_{\text{all}}$ is the space of all functions and $\delta_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[\delta_t \mid S_t = s]$. For a state s, the optimal $h^*(s) = \delta_\pi(s)$, and we recover the $\overline{\text{BE}}$. More generally, we learn a parameterized function that approximates this auxiliary variable h. Letting $\mathcal{H}$ be the space of the parameterized functions for h, the $\overline{\text{GPBE}}$ then projects $\overline{\text{BE}}$ into $\mathcal{H}$, and is defined as:

$$\overline{\text{GPBE}}(\boldsymbol{w}) = \max_{h \in \mathcal{H}} \sum_{s \in S} d(s) \left(2\, \delta_\pi(s)\, h(s) - h(s)^2 \right) . \tag{3}$$

Depending on the choice of $\mathcal{H}$, the $\overline{\text{GPBE}}$ can express a variety of objectives. For a linear function class, we recover the linear $\overline{\text{PBE}}$, and for a highly expressive function class, we recover the (identifiable) $\overline{\text{BE}}$ (Patterson et al., 2022b).

The $\overline{\text{GPBE}}$ can be optimized by taking the gradient of the objective, which results in a saddle point update called *GTD2*, or we can do a gradient correction update, which results in a preferable algorithm called *TDC*. Note that GTD2 and TDC were introduced for the linear setting (Sutton et al., 2009), but the same names are used when generalized to the nonlinear setting (Patterson et al., 2022b), so we follow that convention. TDC has been shown to outperform GTD2 (Ghiassian et al., 2020; White & White, 2016; Patterson et al., 2022b) and has been further extended to include a regularization term, resulting in a better update called TDRC (Ghiassian et al., 2020; Patterson et al., 2022b).

We briefly include the update rule for these three Gradient TD methods, as we will extend them in the following sections. For $\hat{v}$ parameterized by $\boldsymbol{w}$ and $\hat{h}$ parameterized by $\boldsymbol{\theta}$, all methods can be written as jointly updating

$$\begin{aligned} \boldsymbol{w}_{t+1} &\leftarrow \boldsymbol{w}_t + \alpha\, \Delta \boldsymbol{w}_t , \\ \boldsymbol{\theta}_{t+1} &\leftarrow \boldsymbol{\theta}_t + \alpha\, \Delta \boldsymbol{\theta}_t , \end{aligned} \tag{4}$$

where $\alpha \in (0, 1]$ is a step-size hyperparameter, or more generally an optimizer like Adam (Kingma & Ba, 2014) can be used. For GTD2, $\Delta \boldsymbol{w}_t$ is

$$\Delta \boldsymbol{w}_t = -\hat{h}(S_t, \boldsymbol{\theta}_t) \nabla_{\boldsymbol{w}} \delta_t = \hat{h}(S_t, \boldsymbol{\theta}_t) \left(\nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}) - \gamma \nabla_{\boldsymbol{w}} \hat{v}(S_{t+1}, \boldsymbol{w}) \right)$$

The TDC update replaces the term $\hat{h}(S_t, \boldsymbol{\theta}_t) \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w})$ with $\delta_t \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w})$, to get the update

$$\Delta \boldsymbol{w}_t = \delta_t \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}) - \hat{h}(S_t, \boldsymbol{\theta}_t) \nabla_{\boldsymbol{w}} \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$$

This update is called TD with corrections, because the first term is exactly the TD update and the second term acts like a correction to the semi-gradient TD update. This modified update is justified by noting that $h^*(s) = \delta_\pi(s)$ and so replacing the approximation $\hat{h}(S_t, \boldsymbol{\theta}_t)$ with an unbiased sample δ_t instead is sensible. TDC has been shown to converge to the same fixed point as TD and GTD2 in the linear setting (Maei, 2011), and generally has been found to outperform GTD2. Both GTD2 and TDC have the same $\Delta \boldsymbol{\theta}_t$ which can be written as $\Delta \boldsymbol{\theta}_t = \Big(\delta_t - \hat{h}(S_t, \boldsymbol{\theta}_t) \Big) \nabla_{\boldsymbol{\theta}} \hat{h}(S_t, \boldsymbol{\theta}_t)$. TDRC uses the same $\Delta \boldsymbol{w}_t$ as TDC, but regularizes the auxiliary variable:

$$\Delta \boldsymbol{\theta}_t = \Big(\delta_t - \hat{h}(S_t, \boldsymbol{\theta}_t) \Big) \nabla_{\boldsymbol{\theta}} \hat{h}(S_t, \boldsymbol{\theta}_t) - \beta \boldsymbol{\theta}_t .$$

For $\beta = 0$, TDRC is the same as TDC, and as β is increased, h gets pushed closer to zero and TDRC becomes closer to TD. TDRC was found to be strictly better than TDC, even with a fixed $\beta = 1$ across problems (Ghiassian et al., 2020; Patterson et al., 2022b). This improvement was further justified theoretically with a connection to robust Bellman losses (Patterson et al., 2022a), motivating regularization on h.

3 The Generalized PBE(λ) Objective

The basis of $\overline{\text{GPBE}}$ is the 1-step TD error, which means that credit assignment can be slow. Reward information must propagate backward one step at a time through the value function, via bootstrapping. In this section, we extend the $\overline{\text{GPBE}}$ to incorporate multistep credit assignment using the λ-return.

First, let us define our multistep target. The simplest multistep return estimator is the n-step return, defined as

$$G_t^{(n)} \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} \gamma^i R_{t+1+i} + \gamma^n \hat{v}(S_{t+n}, \boldsymbol{w}_t)\,.$$

The λ-return is the exponentially weighted average of all possible n-step returns:

$$G_t^{\lambda} \stackrel{\text{def}}{=} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}\,, \tag{5}$$

where $\lambda \in [0, 1]$. The λ-return is the return target for TD(λ) (Sutton, 1988) and comes with a number of desirable properties: it smoothly interpolates between TD and Monte Carlo methods (a bias-variance trade-off; Kearns & Singh, 2000), reduces variance compared to a single n-step return (Daley et al., 2024b), and imposes a recency heuristic by assigning less weight to temporally distant experiences (Daley et al., 2024a). We denote the error between the λ-return target and the current value estimate by

$$\delta_t^{\lambda} \stackrel{\text{def}}{=} G_t^{\lambda} - \hat{v}(S_t, \boldsymbol{w}_t) = \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_{t+i}\,, \tag{6}$$

and refer to this quantity as the TD(λ) error. We note that in the context of recent works, the TD(λ) error is often referred to as the generalized advantage estimate (GAE; Schulman et al., 2015).

The $\overline{\text{GPBE}}$ is defined using this TD(λ) error. For $\delta_\pi^{\lambda}(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[\delta_t^{\lambda} \mid S_t = s]$, we define the $\overline{\text{BE}}(\lambda)$ analogously to Eq. (1) as

$$\overline{\text{BE}}(\boldsymbol{w}, \lambda) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d(s)\, \delta_\pi^{\lambda}(s)^2\,.$$

Following the earlier derivation of the $\overline{\text{GPBE}}$ in Eq. (3), with the definitions of h and the new $\delta_\pi^{\lambda}(s)$, we can write the $\overline{\text{GPBE}}(\lambda)$ objective as

$$\overline{\text{GPBE}}(\boldsymbol{w}, \lambda) \stackrel{\text{def}}{=} \max_{h \in \mathcal{H}} \sum_{s \in S} d(s) \Big(2\delta_\pi^{\lambda}(s)\, h(s) - h(s)^2 \Big)\,. \tag{7}$$

When $\lambda = 0$, we recover the original $\overline{\text{GPBE}}$ objective of Patterson et al. (2022b). In the absence of function approximation, the $\overline{\text{GPBE}}$ and the $\overline{\text{GPBE}}(\lambda)$ objectives lead to the same solution, v_π, because their fixed points are both v_π. However, when function approximation is introduced, the choice of λ strongly impacts the minimum-error solution. In practice, intermediate λ-values on the interval $(0, 1)$ will balance between solution quality, learning speed, and variance.

4 The Forward-View for Gradient TD(λ) Methods

In this section, we develop several forward-view methods for optimizing the GPBE(λ) under non-linear function approximation. Following the previous convention, we will overload the names

GTD2(λ) and TDC(λ) introduced for the linear setting because we are strictly generalizing them to a broader function class.

GTD2(λ): We derive this algorithm by taking the gradient of Eq. (7) w.r.t to both $\boldsymbol{w}$ and $\boldsymbol{\theta}$.

$$\frac{1}{2}\nabla_{\boldsymbol{w}} \sum_{s\in\mathcal{S}} d(s)\Big(2\,\delta_\pi^\lambda(s)\,h(s) - h(s)^2\Big) = \sum_{s\in\mathcal{S}} d(s)\nabla_{\boldsymbol{w}}\delta_\pi^\lambda(s)\,,$$
$$\frac{1}{2}\nabla_{\boldsymbol{\theta}} \sum_{s\in\mathcal{S}} d(s)\Big(2\,\delta_\pi^\lambda(s)\,h(s) - h(s)^2\Big) = \sum_{s\in\mathcal{S}} d(s)\big(\delta_\pi^\lambda(s) - h(s)\big)\nabla_{\boldsymbol{\theta}}h(s)\,.$$

We get a stochastic gradient descent update by sampling these expressions. For brevity throughout, let $V_t \overset{\text{def}}{=} \hat{v}(S_t, \boldsymbol{w}_t)$ and $H_t \overset{\text{def}}{=} \hat{h}(S_t, \boldsymbol{\theta}_t)$. The resulting update is then

$$\Delta\boldsymbol{w}_t = -H_t\nabla_{\boldsymbol{w}}\delta_t^\lambda\,, \tag{8}$$
$$\Delta\boldsymbol{\theta}_t = (\delta_t^\lambda - H_t)\nabla_\theta H_t\,. \tag{9}$$

GTD2(λ) is a standard saddle-point update and should converge to a local optimum of the $\overline{\text{GPBE}}(\lambda)$ objective.

TDC(λ): For TDC(0), we obtained a gradient correct alternative by adding the term $(\delta_t - h(S_t))\nabla_{\boldsymbol{w}}\hat{v}(S_t, \boldsymbol{w})$ to the GTD2(0) update. This was motivated by the fact that $h(S_t)$ approximates δ_t. We take a similar approach here, adding $(\delta_t^\lambda - H_t)\nabla_{\boldsymbol{w}}\hat{v}(S_t, \boldsymbol{w}_t)$ to the GTD2(λ) update for $\boldsymbol{w}$:

$$\begin{aligned}\Delta\boldsymbol{w}_t &= (\delta_t^\lambda - H_t)\nabla_{\boldsymbol{w}}V_t - H_t\nabla_{\boldsymbol{w}}\delta_t^\lambda \\ &= \delta_t^\lambda\nabla_{\boldsymbol{w}}V_t - H_t\nabla_{\boldsymbol{w}}(V_t + \delta_t^\lambda)\,.\end{aligned} \tag{10}$$

The $\boldsymbol{\theta}$-update remains the same as Eq. (9). The result is the sum of a semi-gradient TD(λ) update and a gradient correction. However, the method is biased, as it assumes that H_t has converged exactly to $\delta_\pi^\lambda(S_t)$. This bias did not impact convergence of TDC in the linear setting, but as yet there is no proof of convergence of TDC in the nonlinear setting. Similarly, it is not yet clear what the ramifications are of using TDC(λ) rather than GTD2(λ), although once again, in our experiments, we find it is better empirically.

TDRC(λ): Finally, we extend the TDRC algorithm, and the extension simply involves adding a regularization penalty with coefficient $\beta \geq 0$ to the update for h:

$$\Delta\boldsymbol{\theta}_t = (\delta_t^\lambda - H_t)\nabla_{\boldsymbol{\theta}}H_t - \beta\boldsymbol{\theta}_t\,. \tag{11}$$

All the methods we derived in this section depend on the forward-view of the λ-return from Eq. (5), which means they need a trajectory of transitions to make an update. This makes these methods appealing when there is a replay buffer to store and sample these trajectories. Further, the trajectories should be on-policy to avoid the need to incorporate importance sampling ratios. It is not difficult to incorporate importance sampling (we include these extensions in Appendix B, but there is the potential for variance issues when using importance sampling. These two criteria motivate why we incorporate these forward-view updates into PPO in the next section.

5 Gradient PPO: Using the Forward-View in Deep RL

In this section, we introduce a new algorithm, called Gradient PPO, that modifies the PPO algorithm (Schulman et al., 2017) to incorporate the forward-view gradient methods derived in the last section.

5.1 Gradient PPO

Proximal Policy Optimization (PPO; Schulman et al., 2017) is a widely used policy-gradient method that learns both a parameterized policy, the actor, and an estimate for the state-value function, the critic. In PPO, the agent alternates between collecting a fixed-length trajectory of interactions and performing batch updates using that trajectory to learn both the policy and the state-value function. We will focus on the critic component of PPO, as that is the part learning the value function, and we will modify it to use the gradient-based methods introduced in Section 4.

PPO updates depend on the Generalized Advantage Estimate (GAE; Schulman et al., 2015), which is identical to the TD(λ) error in Eq. (6). In practice, however, PPO updates must truncate GAE due to the finite length of the collected experience trajectory. Given a trajectory of length T, the truncated GAE can be written as $\delta^{\lambda}_{t:T} = \sum_{i=0}^{T-t-1} (\gamma\lambda)^i \delta_{t+i}$, and we can form an estimate for the λ-return using that truncated GAE as:

$$G^{\lambda}_{t:T} \overset{\text{def}}{=} \hat{v}(S_t, \boldsymbol{w}) + \sum_{k=0}^{T-t-1} (\gamma\lambda)^k \delta_{t+k} \,. \tag{12}$$

The value-function objective for PPO can then be written as follows:

$$L_t(\boldsymbol{w}_t) = \frac{1}{2}\left(\hat{v}(S_t, \boldsymbol{w}_t) - \text{sg}(G^{\lambda}_{t:T})\right)^2 \,, \tag{13}$$

where $\text{sg}(.)$ denotes a stop gradient operation, so the gradient of the objective only accounts for the gradient of $\hat{v}(s_t, \boldsymbol{w}_t)$. PPO typically uses a stale target for $G^{\lambda}_{t:T}$. i.e., the λ-return target is computed once from the collected trajectory and is kept fixed for all the training epochs on that trajectory. Finally, most implementations consider a clipped version of this loss. But, for simplicity, we will drop the clipping part of the objective for our algorithm. Since the clipping, among other heuristics in PPO, is meant to stabilize the updates and prevent large updates, they might not be needed for gradient-based methods that can robustly handle off-policy updates.

We now introduce *Gradient PPO* which changes the critic update for PPO to allow for Gradient TD(λ) updates. Gradient PPO introduces the following three changes.

Modification 1: We change PPO's objective function, Eq. (13), to match the updates in Section 4. We can write a new objective based on TDRC(λ) as follows:

$$L_t(\boldsymbol{w}_t) = \text{sg}\left(\hat{h}(S_t, \boldsymbol{\theta}_t)\right)\delta^{\lambda}_{t:T} - \text{sg}\left(\delta^{\lambda}_{t:T} - \hat{h}(S_t, \boldsymbol{\theta}_t)\right)\hat{v}(S_t, \boldsymbol{w}_t). \tag{14}$$

Modification 2: We introduce an objective function for the auxiliary variable $\hat{h}$, which can be written as:

$$L_t(\boldsymbol{\theta}_t) = -\text{sg}\left(\delta^{\lambda}_{t:T} - \hat{h}(S_t, \boldsymbol{\theta}_t)\right)\hat{h}(S_t, \boldsymbol{\theta}_t) + \frac{\beta}{2}\|\boldsymbol{\theta}_t\|^2. \tag{15}$$

Modification 3: We need to compute the gradient for δ^{λ}_t. As a result, we cannot use a stale target as in Eq. (13). Instead, we need to recompute δ^{λ}_t and its gradient after each update. We do this by sampling sequences from the minibatch instead of sampling independent samples. We then compute a truncated $\delta^{\lambda}_{t:\tau}$ based on the sampled sequences. In this case, the effective truncation for the λ-return is the length of the sequence sampled from a minibatch, τ, rather than the full trajectory length T. Daley & Amato (2019) used a similar approach to incorporate the λ-return with replay buffers. This approach might seem computationally expensive at first since $\boldsymbol{w}$ is used to compute all the values included in $\hat{\delta}^{\lambda}_{t:\tau}$ estimation. However, a nice property of the gradient $\nabla_{\boldsymbol{w}}\hat{\delta}^{\lambda}_{t:\tau}$ is that it can be easily computed recursively as follows:

$$\nabla_{\boldsymbol{w}}\delta^{\lambda}_t = \gamma\lambda\nabla_{\boldsymbol{w}}\delta^{\lambda}_{t+1} + \nabla_{\boldsymbol{w}}\delta_t.$$

Then, given a sequence of length τ, $\nabla_{\boldsymbol{w}}\delta^{\lambda}_t$ and δ^{λ}_t can be estimated using Algorithm 1, where lines in green highlight the additional computations required for Gradient PPO per a minibatch update.

Implementations for Gradient PPO can simply pass the newly defined loss functions, Eq. (14) and Eq. (15), directly to an automatic differentiation. But implementations based on Algorithm 1 might be more efficient as it allows for parallel computations of the values for all states. We also provide a full algorithm for PPO and Gradient PPO in Appendix C.

Algorithm 1 Estimating TDRC(λ) Updates for Gradient PPO

Input: A sequence of states, $s_t, \dots s_{t+\tau}$.
Input: The current weight parameters of the value function, $\boldsymbol{w}$.
For all samples in the sequence, compute $\hat{v}(s_t, \boldsymbol{w})$ and $\nabla_{\boldsymbol{w}} \hat{v}(s_t, \boldsymbol{w})$.
▷ This step is done in parallel by creating a batch of all observations.
for $j = t + \tau - 1, \dots,$ t **do**
  $\delta_j = R_{j+1} + \gamma \hat{v}(s_{j+1}, \boldsymbol{w}) - \hat{v}(s_j, \boldsymbol{w})$
  $\nabla \delta_j = R_{j+1} + \gamma \nabla \hat{v}(s_{j+1}, \boldsymbol{w}) - \nabla \hat{v}(s_j, \boldsymbol{w})$
  $\delta_j^\lambda = \delta_j + \gamma \lambda \delta_{j+1}^\lambda$
  $\nabla \delta_j^\lambda = \nabla \delta_j + \gamma \lambda \nabla \delta_{j+1}^\lambda$
end for

5.2 Empirical Analysis of Gradient PPO

We now evaluate the performance of Gradient PPO across several environments from the MuJoCo Benchmark (Todorov et al., 2012). For Gradient PPO, we performed a hyperparameter sweep for the actor learning rate, the critic learning rate, and λ. For the auxiliary variable h, we used the same learning rate as the critic. We tested each hyperparameter configuration on all environments and repeated the experiments across 5 seeds. Finally, based on the sweep results, we selected a hyperparameter configuration that worked reasonably well across all environments and evaluated it for 30 more seeds. We provide the ranges of values we swept over in Appendix D and the hyperparameters configuration that we will use in all Gradient PPO experiments in Table 4. For PPO, we used the default hyperparameters commonly used for PPO with Mujoco environments (Huang et al., 2022). We provide those default hyperparameters in Table 3.

Figure 1 shows the Gradient PPO and Default PPO results across four MuJoCo environments. In Ant and HalfCheetah, Gradient PPO clearly outperforms PPO. Both algorithms perform similarly in Walker and Hopper.

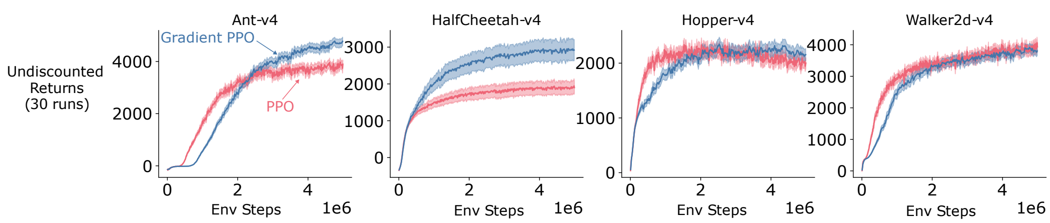

Figure 1: Gradient PPO and PPO evaluated on four MuJoCo environments. The solid lines are the mean performance averaged over 30 seeds, and the shaded area is the standard error.

We also investigated the utility of using TDRC(λ) instead of TDC(λ) and GTD2(λ) to estimate the critic. Figure 2 shows the results with these variations. There is a marked difference in performance, and these results suggest that both gradient corrections and regularization are needed to perform better when using gradient-based methods. This outcome aligns with our discussion in Section 2 and Section 4 about how TDRC has been shown to outperform TDC, which in turn outperforms GTD2.

6 The Backward View for Gradient TD(λ) Methods

The forward-view algorithms we have derived so far have updates that depend on future information, making them unrealizable without the delay introduced by experience replay. Alternatively, we can use eligibility traces via backward-view algorithms that incrementally generate the correct parameter updates on each time step. We now derive the backward view algorithms for optimizing $\overline{\text{GPBE}}(\lambda)$.

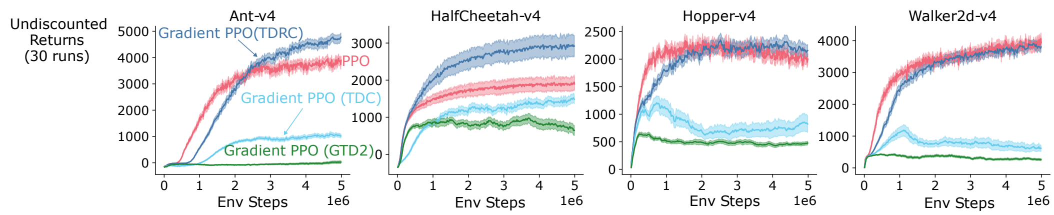

Figure 2: Gradient PPO variations evaluated on 4 Mujoco environments. The solid lines are the mean performance averaged over 30 seeds, and the shaded area is the standard error.

GTD2(λ): As we prove below, the following backward-view updates are equivalent to the forward-view updates given in Eq. 8:

$$\Delta \boldsymbol{w}_t \stackrel{\text{def}}{=} -z_t^h \nabla_{\boldsymbol{w}} \delta_t \,, \tag{16}$$

$$\Delta \boldsymbol{\theta}_t \stackrel{\text{def}}{=} \delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} H_t \,, \tag{17}$$

where

$$z_t^h \stackrel{\text{def}}{=} \gamma\lambda z_{t-1}^h + H_t \,, \tag{18}$$

$$\boldsymbol{z}_t^{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \gamma\lambda \boldsymbol{z}_{t-1}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} H_t \,, \tag{19}$$

for $z_{-1}^h \stackrel{\text{def}}{=} 0$, and $\boldsymbol{z}_{-1}^{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \mathbf{0}$. We show in the following theorem that this backward-view algorithm generates the same total parameter updates as the forward view under standard assumptions.

Theorem 6.1. *Assume the parameters $\boldsymbol{w}$ and $\boldsymbol{\theta}$ do not change during an episode of environment interaction. The forward and backward views of GTD2(λ) are equivalent in the sense that they generate equal total parameter updates:*

$$\sum_{t=0}^{\infty} H_t \nabla_{\boldsymbol{w}} \delta_t^\lambda = \sum_{t=0}^{\infty} z_t^h \nabla_{\boldsymbol{w}} \delta_t \,, \tag{20}$$

$$\sum_{t=0}^{\infty} (\delta_t^\lambda - H_t) \nabla_{\boldsymbol{\theta}} H_t = \sum_{t=0}^{\infty} (\delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} H_t) \,. \tag{21}$$

Proof. See Appendix A. □

Table 2: Forward- and backward-view updates of our three proposed Gradient TD(λ) algorithms for prediction with nonlinear function approximation.

Algorithm	**View**	$\Delta \boldsymbol{w}_t$	$\Delta \boldsymbol{\theta}_t$
B-GTD2(λ)	Forward	$-H_t \nabla_{\boldsymbol{w}} \delta_t^\lambda$	$(\delta_t^\lambda - H_t) \nabla_{\boldsymbol{\theta}} H_t$
	Backward	$-z_t^h \nabla_{\boldsymbol{w}} \delta_t$	$\delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} H_t$
B-TDC(λ)	Forward	$\delta_t^\lambda \nabla_{\boldsymbol{w}} V_t - H_t \nabla_{\boldsymbol{w}} (V_t + \delta_t^\lambda)$	$(\delta_t^\lambda - H_t) \nabla_{\boldsymbol{\theta}} H_t$
	Backward	$\delta_t \boldsymbol{z}_t^{\boldsymbol{w}} - H_t \nabla_{\boldsymbol{w}} V_t - z_t^h \nabla_{\boldsymbol{w}} \delta_t$	$\delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} h_t$
B-TDRC(λ)	Forward	$\delta_t^\lambda \nabla_{\boldsymbol{w}} V_t - H_t \nabla_{\boldsymbol{w}} (V_t + \delta_t^\lambda)$	$(\delta_t^\lambda - H_t) \nabla_{\boldsymbol{\theta}} H_t - \beta \boldsymbol{\theta}_t$
	Backward	$\delta_t \boldsymbol{z}_t^{\boldsymbol{w}} - H_t \nabla_{\boldsymbol{w}} V_t - z_t^h \nabla_{\boldsymbol{w}} \delta_t$	$\delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} H_t - \beta \boldsymbol{\theta}_t$

TDC(λ): Let us slightly rewrite $\Delta \boldsymbol{w}_t$ from Eq. (10) in the following way:

$$\underbrace{\delta_t^\lambda \nabla_{\boldsymbol{w}} V_t}_{\text{TD}(\lambda)} + \underbrace{(-H_t \nabla_{\boldsymbol{w}} V_t)}_{\substack{\text{instantaneous}\\\text{correction}}} + \underbrace{(-H_t \nabla_{\boldsymbol{w}} \delta_t^\lambda)}_{\text{B-GTD2}(\lambda)}. \tag{22}$$

We see that $\Delta \boldsymbol{w}_t$ from Eq. (22) decomposes into three terms: forward-view semi-gradient TD(λ) with off-policy corrections; an instantaneous correction that does not require eligibility traces; and GTD2(λ)'s term for $\Delta \boldsymbol{w}_t$, for which we already derived and proved a backward-view equivalence in Theorem 6.1. As a consequence, we immediately deduce that the backward view for TDC(λ) is

$$\Delta \boldsymbol{w}_t \stackrel{\text{def}}{=} \delta_t \boldsymbol{z}_t^{\boldsymbol{w}} - H_t \nabla_{\boldsymbol{w}} V_t - z_t^{\boldsymbol{w}} \nabla_{\boldsymbol{w}} \delta_t, \tag{23}$$

where

$$\boldsymbol{z}_t^{\boldsymbol{w}} \stackrel{\text{def}}{=} \gamma\lambda \boldsymbol{z}_{t-1}^{\boldsymbol{w}} + \nabla_{\boldsymbol{w}} V_t, \tag{24}$$

and z_t^h is the same as before in Eq. (18). $\Delta \boldsymbol{\theta}_t$ is generated by Eq. (17).

TDRC(λ): Likewise, the regularized backward-view $\boldsymbol{\theta}$ update is

$$\Delta \boldsymbol{\theta}_t \stackrel{\text{def}}{=} \delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} H_t - \beta \boldsymbol{\theta}_t, \tag{25}$$

where $\boldsymbol{z}_t^{\boldsymbol{\theta}}$ is once again generated by Eq. (19). Table 2 summarizes the forward view and the backward view for all the algorithms introduced. We highlighted the update components that arise from directly taking the gradient of $\overline{\text{GPBE}}(\lambda)$ in green, the gradient correction components in blue, and the regularization component in orange.

7 QRC(λ): Using the Backward-view in Deep RL

In this section, we extend the backward-view methods to action values and present three control algorithms based on three backward-view updates presented earlier. Since these algorithms are based on the backward view, they can make immediate updates without delay. Hence, they can work effectively in settings where it is prohibitive to have a large experience replay buffer, i.e., on-edge devices and mobile robots. Additionally, unlike forward-view methods, which require us to present a truncated version of the updates, backward-view methods do not have this limitation.

7.1 QRC(λ)

Extending the backward-view algorithms to action values is straightforward. Here, we present the extensions to Q(λ), but similar extensions can be done to other action-value methods, such as SARSA(λ). Note that similar changes can be made to action-value methods using the forward view.

Consider an action-value network parameterized by $\boldsymbol{w}$, and write the TD error as:

$$\delta_t = R_{t+1} + \gamma \max_{a' \in \mathcal{A}} \hat{q}(S_{t+1}, a', \boldsymbol{w}_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t).$$

The gradient of the TD error becomes the following:

$$\nabla_{\boldsymbol{w}_t} \delta_t = \gamma \nabla_{\boldsymbol{w}_t} \left(\max_{a' \in \mathcal{A}} \hat{q}(S_{t+1}, a', \boldsymbol{w}_t) \right) - \nabla_{\boldsymbol{w}_t} \hat{q}(S_t, A_t, \boldsymbol{w}_t).$$

The auxiliary function for h is now predicting a function of both the states and actions: $h_t \stackrel{\text{def}}{=} h(s_t, a_t, \boldsymbol{\theta}_t)$. Using these modifications, we can now write the updates for the control variant of

TDRC(λ), which we refer to as QRC(λ):

$$\begin{aligned}
\boldsymbol{z}_t^{\boldsymbol{w}} &= \gamma\lambda \boldsymbol{z}_{t-1}^{\boldsymbol{w}} + \nabla_{\boldsymbol{w}_t}\hat{q}(S_t, A_t, \boldsymbol{w}_t) \\
z_t^h &= \gamma\lambda z_{t-1}^h + H_t \\
\boldsymbol{z}_t^{\boldsymbol{\theta}} &= \gamma\lambda \boldsymbol{z}_{t-1}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} H_t \\
\Delta \boldsymbol{w}_t &= \delta_t \boldsymbol{z}_t^{\boldsymbol{w}} - H_t \nabla_{\boldsymbol{w}}\hat{q}(S_t, A_t, \boldsymbol{w}_t) - z_t^h \nabla_{\boldsymbol{w}}\delta_t \\
\Delta \boldsymbol{\theta}_t &= \delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} H_t - \beta \boldsymbol{\theta}_t
\end{aligned} \tag{26}$$

We can modify these updates to get QC(λ), an update based on TDC(λ), by simply setting $\beta = 0$. We can also get GQ(λ), an update based on GTD2(λ) by setting $\beta = 0$ and removing the gradient correction term (see Table 2). Finally, we follow Watkins' Q(λ) in that we decay the traces as described in the previous equations when a greedy action is selected and reset the traces to zero when a non-greedy action is selected (Watkins, 1989).

7.2 Empirical Analysis of QRC(λ)

We evaluated the performance of QRC(λ) across all the environments from the MinAtar benchmark (Young & Tian, 2019). We compared the performance with Watkin's Q(λ) (Watkins, 1989) and StreamQ algorithm (Elsayed et al., 2024), a recent algorithm combining Q(λ) with a new optimizer and an initialization scheme for better performance in streaming settings.

For Q(λ) and QRC(λ), we used SGD and performed a hyperparameter sweep for different values for the step size and λ. We tested each hyperparameter configuration in all environments and across 5 seeds. We then selected the hyperparameter configuration that worked well across all environments, and we evaluated it for 30 more seeds in all environments. We provide the ranges and the final hyperparameters we used in Appendix E. For StreamQ, we did not do a hyperparameter sweep, as the paper claimed that their algorithm is robust to hyperparameters and does not need a sweep over them. Hence, we report the results of the StreamQ algorithm based on running their available code with its default hyperparameters across 30 seeds in all environments. Figure 3 shows the performance of all three algorithms across the 5 MinAtar environments, and in all environments, QRC(λ) outperforms both StreamQ and Q(λ).[2]

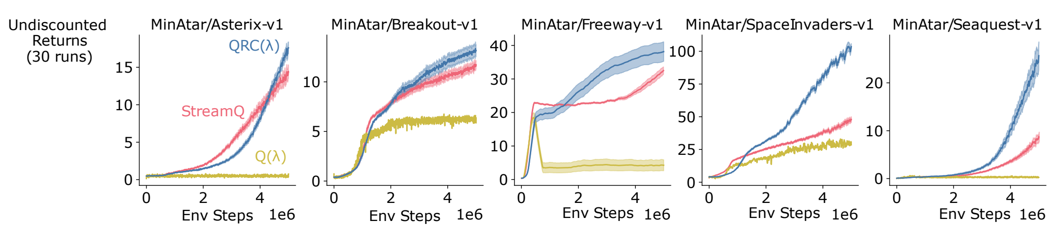

Figure 3: QRC(λ), Q(λ) and StreamQ algorithms evaluated on the five MinAtar environments. The solid lines are the mean performance averaged over 30 seeds, and the shaded regions are the corresponding standard errors.

We evaluated the other two gradient-based algorithms, QC(λ) and GQ2(λ). Figure 4 shows the results of this evaluation. The results are consistent with forward-view results in Section 5 in that having both the gradient correction and the regularization is needed for better performance. However, here the regularization is not as critical as it was for Gradient PPO.

[2]Note that StreamQ results are lower than what was reported by Elsayed et al. (2024) on SpaceInvaders and Seaquest. However, we obtained those results from their publicly available code without any modifications.

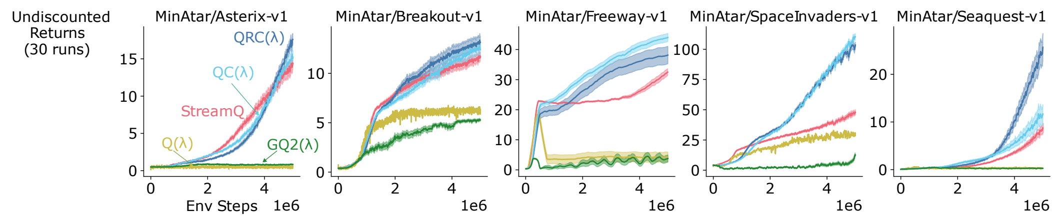

Figure 4: All gradient-based backward view algorithms evaluated on the 5 MinAtar environments. The solid lines are the mean performance averaged over 30 seeds, and the shaded regions are the corresponding standard errors.

8 Conclusion

We proposed the $\overline{\text{GPBE}}(\lambda)$ objective, a multistep generalization of the Generalized Projected Bellman Error (Patterson et al., 2022b) based on the λ-return. We derived three algorithms for optimizing the new objective both in the forward view and in the backward view. Of the three algorithms we developed, we showed that TDRC(λ) is stable, fast, and convergent to a high-quality solution. We introduced two Deep RL algorithms that use the newly derived update rules, and we showed that our new algorithms outperform both PPO with a buffer and streaming algorithms without replay buffers.

A Proof of Theorem 6.1

Theorem 6.1. *Assume the parameters $\boldsymbol{w}$ and $\boldsymbol{\theta}$ do not change during an episode of environment interaction. The forward and backward views of GTD2(λ) are equivalent in the sense that they generate equal total parameter updates:*

$$\sum_{t=0}^{\infty} H_t \nabla_{\boldsymbol{w}} \delta_t^\lambda = \sum_{t=0}^{\infty} z_t^h \nabla_{\boldsymbol{w}} \delta_t \,, \tag{20}$$

$$\sum_{t=0}^{\infty} (\delta_t^\lambda - H_t) \nabla_{\boldsymbol{\theta}} H_t = \sum_{t=0}^{\infty} (\delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} H_t) \,. \tag{21}$$

In the proof below, we added importance sampling for generality.

Proof. We start by showing Eq. (20) holds. Note that

$$H_t \nabla_{\boldsymbol{w}} \hat{\delta}_t^\lambda = H_t \rho_t \nabla_{\boldsymbol{w}} \delta_t + H_t \gamma\lambda \rho_t \rho_{t+1} \nabla_{\boldsymbol{w}} \delta_{t+1} + H_t (\gamma\lambda)^2 \rho_t \rho_{t+1} \rho_{t+2} \nabla_{\boldsymbol{w}} \delta_{t+2} \dots . \tag{27}$$

The total sum of these forward-view contributions is therefore

$$\begin{aligned}
\sum_{t=0}^{\infty} H_t \nabla_{\boldsymbol{w}} \delta_t^\lambda &= (H_0 \rho_0 \nabla_{\boldsymbol{w}} \delta_0 + H_0 \gamma\lambda \rho_0 \rho_1 \nabla_{\boldsymbol{w}} \delta_1 + \dots) + (H_1 \rho_1 \nabla_{\boldsymbol{w}} \delta_1 + H_1 \gamma\lambda \rho_1 \rho_2 \nabla_{\boldsymbol{w}} \delta_2 + \dots) + \dots \\
&= (H_0 \rho_0) \nabla_{\boldsymbol{w}} \delta_0 + (H_0 \gamma\lambda \rho_0 \rho_1 + H_1 \rho_1) \nabla_{\boldsymbol{w}} \delta_1 + \dots \qquad (28) \\
&= z_0^h \nabla_{\boldsymbol{w}} \delta_0 + z_1^h \nabla_{\boldsymbol{w}} \delta_1 + \dots \qquad (29) \\
&= \sum_{t=0}^{\infty} z_t^h \nabla_{\boldsymbol{w}} \delta_t, \qquad (30)
\end{aligned}$$

which proves Eq. (20). Next, consider Eq. (21). Notice that the equality holds if and only if

$$\sum_{t=0}^{\infty} \hat{\delta}_t^\lambda \nabla_{\boldsymbol{\theta}} H_t = \sum_{t=0}^{\infty} \delta_t z_t^{\boldsymbol{\theta}}, \tag{31}$$

and further note that

$$\hat{\delta}_t^\lambda \nabla_{\boldsymbol{\theta}} H_t = \rho_t \delta_t \nabla_{\boldsymbol{\theta}} H_t + \gamma\lambda \rho_t \rho_{t+1} \delta_{t+1} \nabla_{\boldsymbol{\theta}} H_t + (\gamma\lambda)^2 \rho_t \rho_{t+1} \rho_{t+2} \delta_{t+2} \nabla_{\boldsymbol{\theta}} H_t + \dots. \tag{32}$$

The total sum of these forward-view contributions is therefore

$$\sum_{t=0}^{\infty} \delta_t^\lambda \nabla_{\boldsymbol{\theta}} H_t = (\rho_0 \delta_0 \nabla_{\boldsymbol{\theta}} H_0 + \gamma\lambda \rho_0 \rho_1 \delta_1 \nabla_{\boldsymbol{\theta}} H_0 + \dots) + (\rho_1 \delta_1 \nabla_{\boldsymbol{\theta}} H_1 + \gamma\lambda \rho_1 \rho_2 \delta_2 \nabla_{\boldsymbol{\theta}} H_1 + \dots) + \dots \tag{33}$$

$$= \delta_0 (\rho_0 \nabla_{\boldsymbol{\theta}} H_0) + \delta_1 (\gamma\lambda \rho_0 \rho_1 \nabla_{\boldsymbol{\theta}} H_0 + \rho_1 \nabla_{\boldsymbol{\theta}} H_1) + \dots \tag{34}$$

$$= \delta_0 z_0^{\boldsymbol{\theta}} + \delta_1 z_1^{\boldsymbol{\theta}} + \dots \tag{35}$$

$$= \sum_{t=0}^{\infty} \delta_t z_t^{\boldsymbol{\theta}}, \tag{36}$$

which establishes Eq. (31) to prove Eq. (21) and complete the proof. □

References

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning*. 1995.

Bo Dai, Niao He, Yunpeng Pan, Byron Boots, and Le Song. Learning from conditional distributions via dual embeddings. In *Artificial Intelligence and Statistics*, 2017.

Brett Daley and Christopher Amato. Reconciling λ-returns with experience replay. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Brett Daley, Marlos C. Machado, and Martha White. Demystifying the recency heuristic in temporal-difference learning. *Reinforcement Learning Journal (RLJ)*, 2024a.

Brett Daley, Martha White, and Marlos C. Machado. Averaging n-step returns reduce variance in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2024b.

Mohamed Elsayed, Gautham Vasan, and A Rupam Mahmood. Streaming deep reinforcement learning finally works. *arXiv preprint arXiv:2410.14606*, 2024.

Sina Ghiassian, Andrew Patterson, Shivam Garg, Dhawal Gupta, Adam White, and Martha White. Gradient temporal-difference learning with regularized corrections. In *International Conference on Machine Learning*, 2020.

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022.

Michael J. Kearns and Satinder Singh. Bias-variance error bounds for temporal difference updates. In *Conference on Learning Theory (COLT)*, 2000.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Hamid Maei. *Gradient temporal-difference learning algorithms*. PhD thesis, 2011.

Hamid Maei and Richard S. Sutton. GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Conference on Artificial General Intelligence (AGI)*, 2010.

Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. *Advances in Neural Information Processing Systems*, 2009.

Andrew Patterson, Victor Liao, and Martha White. Robust losses for learning value functions. *Transactions on pattern analysis and machine intelligence*, 2022a.

Andrew Patterson, Adam White, and Martha White. A generalized projected Bellman error for off-policy value estimation in reinforcement learning. *Journal of Machine Learning Research*, 2022b.

Doina Precup, Richard S Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. In *International Conference on Machine Learning*, 2000.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 1988.

Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning*, 2009.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

John Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. *Advances in Neural Information Processing Systems*, 1996.

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

Adam White and Martha White. Investigating practical linear temporal difference learning. *arXiv preprint arXiv:1602.08771*, 2016.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Supplementary Materials

The following content was not necessarily subject to peer review.

B Gradient TD (λ) with Importance Sampling Correction

We now discuss the modifications needed when the experiences (S_t, A_t, R_t, S_{t+1}) are collected by a behaviour policy b rather than the target policy π. Letting $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$ be the importance sampling ratio at time t, we can scale the TD error by this factor to form a bias-corrected TD error $\hat{\delta}_t \overset{\text{def}}{=} \rho_t \delta_t$, since $\mathbb{E}_b[\rho_t \delta_t | S_t = s] = \mathbb{E}_\pi[\delta_t | S_t = s] = \boldsymbol{\delta}(s)$ (Precup et al., 2000). By induction, it follows that the bias-corrected TD(λ) error is

$$\hat{\delta}_t^\lambda \overset{\text{def}}{=} \sum_{i=0}^{\infty} (\gamma\lambda)^i \left(\prod_{j=0}^{i} \rho_{t+j} \right) \delta_{t+i} = \rho_t(\gamma\lambda \hat{\delta}_{t+1}^\lambda + \delta_t). \tag{37}$$

The backward view traces will then be defined as follows:

$$z_t^h \overset{\text{def}}{=} \rho_t(\gamma\lambda z_{t-1}^h + h_t)\,, \tag{38}$$

$$\boldsymbol{z}_t^{\boldsymbol{\theta}} \overset{\text{def}}{=} \rho_t(\gamma\lambda \boldsymbol{z}_{t-1}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} h_t)\,, \tag{39}$$

C Gradient PPO

The full PPO algorithm is expanded in algorithm 2 and the full Gradient PPO with B-TDRC updates algorithm is in algorithm 3.

D Experimental Details of Gradient PPO

Table 3 contains all the hyperparameters used for PPO experiments. Table 4 contains all the hyperparameters used for Gradient PPO experiments, and Table 5 shows the ranges we used for the sweep.

E Experimental Details of MinAtar

Table 6 shows the final hyperparameters used for QRC(λ), QC(λ) and GQ2(λ) in all MinAtar environments and Table 7 shows the ranges used for the sweep.

Algorithm 2 PPO with Advantage Estimates

Input: a differentiable policy parametrization $\pi(a|\mathbf{o}, \boldsymbol{\theta})$
Input: a differentiable state-value function parametrization $\hat{v}(\mathbf{o}, \mathbf{w})$
Algorithm parameters: learning rate α, rollout length τ, mini-batche size n, number of epochs k, value coefficient c_1, entropy coefficient c_2, clip coefficient ϵ, max gradient norm c.
for iteration $= 1, 2, \cdots, \tau$ **do**
  Run $\pi_{\text{old}}(a|\mathbf{o}, \boldsymbol{\theta}_{\text{old}})$ for τ timesteps and save transitions of $\langle \mathbf{o}_t, A_{t+1}, R_{t+1}, \log \pi_{old}(A_{t+1}|\mathbf{o}_t, \boldsymbol{\theta}_{\text{old}}), \hat{v}(\mathbf{o}_t, \boldsymbol{\theta}_{\text{old}})\rangle, \ldots,$
$\langle \mathbf{o}_{t+\tau-1}, A_{t+\tau}, R_{t+\tau}, \log \pi_{old}(A_{t+\tau}|\mathbf{o}_{t+\tau-1}, \boldsymbol{\theta}_{\text{old}}), \hat{v}(\mathbf{o}_{t+\tau-1}, \mathbf{w}_{\text{old}})\rangle$
  Calculate $\hat{v}(\mathbf{o}_{t+\tau}, \mathbf{w}_{\text{old}})$ ▷ For bootstrapping
  Set $\hat{A}^{(\gamma,\lambda)}_{t+\tau} = 0$

  for $j = t + \tau - 1, \ldots,$ t **do**
    $\delta_j = R_{j+1} + \gamma \hat{v}(\mathbf{o}_{j+1}, \mathbf{w}_{\text{old}}) - \hat{v}(\mathbf{o}_j, \mathbf{w}_{\text{old}})$
    $\hat{A}^{\lambda}_j = \delta_j + \gamma\lambda \hat{A}^{\lambda}_{j+1}$
    $\hat{G}^{\lambda}_j = \hat{A}^{\lambda}_j + \hat{v}(\mathbf{o}_j, \mathbf{w}_{\text{old}})$
  end for
    Construct a batch of τ transitions, each transition is:
$\langle \mathbf{o}_i, A_{i+1}, R_{i+1}, \log \pi_{old}(A_{i+1}|\mathbf{o}_i, \boldsymbol{\theta}_{\text{old}}), \hat{v}(o_i, \mathbf{w}_{\text{old}}), \hat{G}^{\lambda}_{i,\mathbf{w}_{\text{old}}}, \hat{A}^{(\gamma,\lambda)}_{i,\mathbf{w}_{\text{old}}}\rangle$
    for epoch = $1, \ldots,$ k **do** ▷ **Learning**
      Shuffle the transitions
      Number of minibatches, $m = \tau/n$
      Divide the data into m mini-batches of size n
      for mini-batch $= 1, \ldots, m$ **do**
        Calculate: $\log \pi_{new}(a|o, \boldsymbol{\theta}_{\text{new}})$, $\hat{v}(o, \mathbf{w}_{\text{new}})$ for samples in the mini-batch.
        Normalize $\hat{A}^{\lambda}$ estimates.
        Policy objective: $L_{\text{p}} = -\frac{1}{n}\sum_{j=1}^{n} \min(r_j \hat{A}_{j,\mathbf{w}_{\text{old}}}, \text{clip}_{\epsilon}(r_j)\hat{A}_{j,\mathbf{w}_{\text{old}}})$
        where $r_j = \frac{\pi(a_j|s_j,\boldsymbol{\theta}_{\text{new}})}{\pi(a_j|s_j,\boldsymbol{\theta}_{\text{old}})}$, and $\text{clip}_{\epsilon}(r_j) = \text{clip}(r_j, 1-\epsilon, 1+\epsilon)$
        Value objective: $L_{\text{v}} = \frac{1}{n}\sum_{j=1}^{n} \max((\hat{v}(\mathbf{o}_j, \mathbf{w}_{\text{new}}) - \hat{G}^{\lambda}_{j,\mathbf{w}_{\text{old}}})^2, (\text{clip}_{\epsilon}(\hat{v}) - \hat{G}^{\lambda}_j)^2)$,
        where $\text{clip}_{\epsilon}(\hat{v}) = \text{clip}(\hat{v}(\mathbf{o}_j, \mathbf{w}_{\text{new}}), 1-\epsilon, 1+\epsilon)$
        Calculate the entropy of the policy: $\frac{1}{n}\sum_{j=1}^{n} S(\pi(\mathbf{o}_j, \boldsymbol{\theta}_{\text{new}}))$
        Calculate the total loss: $L = L_{\text{p}} + c_1 L_{\text{v}} - c_2 S(\pi(\mathbf{s}_t, \boldsymbol{\theta}_{\text{new}}))$
        Calculate the gradient $\hat{g}$
        if $\|\hat{g}\| > c$ **then**
          $\hat{g} \leftarrow \frac{c}{\|\hat{g}\|}\hat{g}$
        end if
        Update the parameters using the gradient to minimize the loss function.
      end for
    end for
  end for

Algorithm 3 PPO with TDRC(λ) (Gradient PPO)

Input: a differentiable policy parametrization $\pi(a|\mathbf{o}, \boldsymbol{\theta})$
Input: a differentiable state-value function parametrization $\hat{v}(\mathbf{o}, \mathbf{w})$
Input: a differentiable auxiliary function parametrization $\hat{h}(\mathbf{o}, \boldsymbol{\theta}_h)$
Algorithm parameters: learning rate α, rollout length τ, mini-batche size n, number of epochs k, value coefficient c_1, entropy coefficient c_2, clip coefficient ϵ, max gradient norm c,Truncation Length T, h learning rate α_h, regularization coefficient $\beta = 1$
for iteration $= 1, 2, \cdots, \tau$ **do**
  Run $\pi_{\text{old}}(a|\mathbf{o}, \boldsymbol{\theta})$ for τ timesteps and save transitions of
  $\langle \mathbf{o}_t, A_{t+1}, R_{t+1}, \log \pi_{old}(A_{t+1}|\mathbf{o}_t, \boldsymbol{\theta}_{\text{old}}), \hat{v}(\mathbf{o}_t, \boldsymbol{\theta}_{\text{old}})\rangle, \ldots,$
  $\langle \mathbf{o}_{t+\tau-1}, A_{t+\tau}, R_{t+\tau}, \log \pi_{old}(A_{t+\tau}|\mathbf{o}_{t+\tau-1}, \boldsymbol{\theta}_{\text{old}}), \hat{v}(\mathbf{o}_{t+\tau-1}, \mathbf{w}_{\text{old}})\rangle$
  Calculate $\hat{v}(\mathbf{o}_{t+\tau}, \mathbf{w}_{\text{old}})$ ▷ For bootstrapping
  Set $\hat{A}_{t+\tau}^{(\gamma,\lambda)} = 0$

  Construct a batch of $\frac{\tau}{T}$ sequences, each sequence is:
  $\langle \mathbf{o}_i, A_{i+1}, R_{i+1}, \log \pi_{old}(A_{i+1}|\mathbf{o}_i, \boldsymbol{\theta}_{\text{old}}), \hat{v}(o_i, \mathbf{w}_{\text{old}})\rangle, \ldots$
  $\langle \mathbf{o}_{i+T}, A_{i+T+1}, R_{i+T+1}, \log \pi_{old}(A_{i+T+1}|\mathbf{o}_{i+T}, \boldsymbol{\theta}_{\text{old}}), \hat{v}(o_{i+T}, \mathbf{w}_{\text{old}})\rangle$
  for epoch = 1, ..., k **do** ▷ **Learning**
    Shuffle the sequences
    Number of minibatches, $m = \tau/(n * T)$
    Divide the data into m mini-batches of size n
    for mini-batch $= 1, \ldots, m$ **do**
      Compute the value gradients for all samples.
      for $j = t + \tau - 1, \ldots,$ t **do** ▷ This loop can be parallelized over the sequences.
        $\delta_j = R_{j+1} + \gamma\hat{v}(\mathbf{o}_{j+1}, \mathbf{w}_{\text{new}}) - \hat{v}(\mathbf{o}_j, \mathbf{w}_{\text{new}})$
        $\nabla\delta_{j_{\mathbf{w}_{\text{new}}}} = R_{j+1} + \gamma\nabla\hat{v}(\mathbf{o}_{j+1}, \mathbf{w}_{\text{new}}) - \nabla\hat{v}(\mathbf{o}_j, \mathbf{w}_{\text{new}})$
        $\delta_j^\lambda = \delta_j + \gamma\lambda\delta_{j+1}^\lambda$
        $\nabla\delta_j^\lambda = \nabla\delta_j + \gamma\lambda\nabla\delta_{j+1}^\lambda$
      end for
      Calculate: $\log \pi_{new}(a|o, \boldsymbol{\theta}_{\text{new}})$, for samples in the mini-batch.
      Policy objective: $L_{\text{p}} = -\frac{1}{n}\sum_{j=1}^{n} \min(r_j\hat{A}_{j,\mathbf{w}_{\text{old}}}, \text{clip}_\epsilon(r_j)\hat{A}_{j,\mathbf{w}_{\text{old}}})$
      where $r_j = \frac{\pi(a_j|s_j, \boldsymbol{\theta}_{\text{new}})}{\pi(a_j|s_j, \boldsymbol{\theta}_{\text{old}})}$, and $\text{clip}_\epsilon(r_j) = \text{clip}(r_j, 1-\epsilon, 1+\epsilon)$
      Calculate the entropy of the policy: $\frac{1}{n}\sum_{j=1}^{n} S(\pi(\mathbf{o}_j, \boldsymbol{\theta}_{\text{new}}))$
      Calculate the total loss: $L = L_{\text{p}} - c_2 S(\pi(\mathbf{s}_t, \boldsymbol{\theta}_{\text{new}}))$
      Calculate the gradient $\hat{g}$
      if $\|\hat{g}\| > c$ **then**
        $\hat{g} \leftarrow \frac{c}{\|\hat{g}\|}\hat{g}$
      end if
      Update the policy using the gradient to minimize the loss function.
      Update Value parameters using the following update:
      $\delta_t^\lambda \nabla_{\boldsymbol{w}} v_t - h_t \nabla_{\boldsymbol{w}}(v_t + \delta_t^\lambda)\ (\delta_t^\lambda - h_t)\nabla_{\boldsymbol{\theta}} h_t - \beta\boldsymbol{\theta}_{h,t}$

      Update h parameters using the following update:
      $(\delta_t^\lambda - h_t)\nabla_{\boldsymbol{\theta}} h_t - \beta\boldsymbol{\theta}_{h,t}$
    end for
  end for
end for

Name	Default Value
Policy Network	(64, tanh, 64, tanh, Linear) + Standard deviation variable
Value Network	(64, tanh, 64, tanh, Linear)
Buffer size	2048
Num epochs	4
Mini-batch size	256
GAE, λ	0.95
Discount factor, γ	0.99
Clip parameter	0.2
Input Normalization	True
Advantage Normalization	True
Value function loss clipping	True
Max Gradient Norm	0.5
Optimizer	Adam
Actor step size	0.0003
Critic step size	0.0003
Optimizer ϵ	1×10^{-5}

Table 3: PPO Hyperparameters and their default values

Name	Default Value
Policy Network	(64, tanh, 64, tanh, Linear) + Standard deviation variable
Value Network	(64, tanh, 64, tanh, Linear)
Buffer size	2048
Num epochs	4
Mini-batch size	256 (split into 8 sequences of length 32)
λ	0.8
Discount factor, γ	0.99
Clip parameter	0.2
Input Normalization	True
Advantage Normalization	True
Max Gradient Norm	0.5
Optimizer	Adam
Actor step size	0.0003
Critic step size	0.003
h step size	0.003
regularization coef, β	1.0
Optimizer ϵ	1×10^{-5}

Table 4: Gradient PPO Hyperparameters and their default values

Name	Sweep Range
λ	$[0.7, 0.8, 0.9, 0.95]$
Actor step size	$[0.001, 0.003, 0.0001, 0.0003, 0.00001, 0.00003]$
Critic step size	$[0.001, 0.003, 0.0001, 0.0003, 0.00001, 0.00003]$
regularization coef, β	$[1.0, 0.0]$

Table 5: Hyperparameter ranges that were used for the sweep experiments for Gradient PPO.

Name	Default Value
λ	0.8
Input Normalization	True
Optimizer	SGD
step size	0.0001
h step size	0.001
regularization coef, β	1.0/0.0
start exploration ϵ,	1.0
end exploration ϵ,	0.01

Table 6: Hyperparameters for MinAtar

Name	Default Value
λ	[0.7,0.8,0.9,0.95]
Optimizer	SGD
step size	[0.001,0.0001,0.00001,0.000001]
h step scale	[1.0,0.1]
regularization coef, β	[1.0,0.0]

Table 7: Hyperparameters ranges for MinAtar sweep