# Deep Reinforcement Learning with Gradient Eligibility Traces

**Esraa Elelimy, Brett Daley, Andrew Patterson, Marlos C. Machado, Adam White, Martha White**

**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 support multistep credit assignment based on the $\lambda$-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. [a]

[a] Code available at https://github.com/esraaelelimy/gtd_algos

## Contribution(s)

1. We extend the $\overline{\text{GPBE}}$ to incorporate multistep credit assignment based on $\lambda$-returns, defining a new objective, the $\overline{\text{GPBE}}(\lambda)$ (Section 3).
   **Context:** Patterson et al. (2022) introduced the $\overline{\text{GPBE}}$, which unifies and generalizes previously known objectives for value estimation. However, it was only defined for the one-step TD error.
2. We derive three Gradient TD algorithms that optimize our proposed objective. We derive both the forward view with the $\lambda$-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. (2022) extended these methods to non-linear function approximation without a need for second-order information. However, it was limited to the one-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($\lambda$), 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($\lambda$) is the first backward-view algorithm that uses Gradient TD methods in the streaming deep RL.

# Deep Reinforcement Learning with Gradient Eligibility Traces

**Esraa Elelimy**[1,2,†], **Brett Daley**[1,2,†], **Andrew Patterson**[1,2], **Marlos C. Machado**[1,2,3], **Adam White**[1,2,3], **Martha White**[1,2,3]
{elelimy, brett.daley, ap3, machado, amw8, whitem}@ualberta.ca

[1]**Department of Computing Science, University of Alberta, Canada**
[2]**Alberta Machine Intelligence Institute (Amii)**
[3]**Canada CIFAR AI Chair**

[†] Equal contribution.

## 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 support multistep credit assignment based on the $\lambda$-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]

## 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, 1997) and under off-policy sampling (Baird, 1995), both of which frequently arise in modern deep RL settings.

There have been significant advances towards deriving sound off-policy TD algorithms. 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, which produces a different solution from 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

[1]Code available at https://github.com/esraaelelimy/gtd_algos

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

| | **Linear Function Approximation** | | **Nonlinear Function Approximation** | |
|---|---|---|---|---|
| **Objective** | **one-step** | **$\lambda$-return** | **1-step** | **$\lambda$-return** |
| PBE | (Sutton et al., 2009) | (Maei & Sutton, 2010) | (Maei et al., 2009) | Our paper |
| GPBE | (Patterson et al., 2022) | Our paper | (Patterson et al., 2022) | Our paper |

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). The Generalized $\overline{\text{PBE}}$ ($\overline{\text{GPBE}}$), 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 (Patterson et al., 2022). This generalized objective was further extended to allow for robust losses in the Bellman error (Patterson et al., 2023) and is a promising avenue for the development of sound value-estimation algorithms. The $\overline{\text{GPBE}}$ and robust extensions, however, has not been extended to include multi-step updates.

In this paper, we extend the $\overline{\text{GPBE}}$ to incorporate multistep credit assignment using $\lambda$-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., 2022), 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 show that Gradient PPO significantly outperforms PPO in two MuJoCo environments and is comparable in two others. We also introduce QRC($\lambda$), which uses backward-view (i.e., eligibility trace updates) and is suitable for online streaming settings.[2] We show that QRC($\lambda$) is significantly better in all MinAtar environments than StreamQ (Elsayed et al., 2024), a recent algorithm combining Q($\lambda$) 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, as was concluded for $\overline{\text{GPBE}}(0)$ (Ghiassian et al., 2020; Patterson et al., 2022), we find that a variant based on regularized corrections called TDRC consistently outperforms the other variants. This work provides a clear way to incorporate gradient TD methods with eligibility traces into deep RL methods and offers two new promising algorithms that perform well in practice.

## 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 $t$ with 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 that can be used to learn this value function is the mean squared Bellman error ($\overline{\text{BE}}$):

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

[2] A setting motivated by hardware limitations where we replay buffers are not used and updates are made one sample at a time. See Elsayed et al. (2024).

where $d$ is the state distribution[3] and $\delta$ is the TD error for a transition $(S, A, S', R)$. The $\delta$ 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 $A = a$ and sum over $(s, a)$ instead of $s$ in Eq. (1), but for simplicity of exposition, we only show the objectives for $\hat{v}$. We cannot generally reach zero $\overline{\text{BE}}$, unless the true values are representable by our parameterized function class for all states with nonzero weight. The $\overline{\text{BE}}$ objective is difficult to optimize, due to the double sampling and identifiability issues (Sutton & Barto, 2018), and we instead consider a more practical objective called the $\overline{\text{GPBE}}$.

The $\overline{\text{GPBE}}$ objective generalizes and unifies several objectives and extends Gradient TD methods to nonlinear function approximation (Patterson et al., 2022). The $\overline{\text{GPBE}}$ builds on prior work (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 can 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:

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

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., 2022).

The $\overline{\text{GPBE}}$ can be optimized by taking the gradient of Eq. (3), which results in a saddle-point update called *GTD2*. Alternatively, we can do a gradient correction update, which results in the emperically 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., 2022), so we follow that convention. TDC has been shown to outperform GTD2 (Ghiassian et al., 2020; White & White, 2016; Patterson et al., 2022) and has been further extended to include a regularization term, resulting in a better update called TDRC (Ghiassian et al., 2020; Patterson et al., 2022).

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

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

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

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

[3]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.

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 motivated by noting that $h^*(s) = \delta_\pi(s)$, and so replacing the approximation $\hat{h}(S_t, \boldsymbol{\theta}_t)$ with an unbiased sample $\delta_t$ instead is sensible. TDC has been shown to converge to the same fixed point as TD and GTD2 in the linear setting (Maei, 2011). 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:

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

where $\beta \in [0, \infty)$. For $\beta = 0$, TDRC is the same as TDC. As $\beta$ 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 several problems (Ghiassian et al., 2020; Patterson et al., 2022). This improvement was further justified theoretically with a connection to robust Bellman losses (Patterson et al., 2023), motivating regularization on $h$.

## 3 The Generalized PBE($\lambda$) 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 $\lambda$-return.

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

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

The $\lambda$-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 $\lambda$-return is the return target for TD($\lambda$) (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 $\lambda$-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($\lambda$) error. We note that in the context of recent works, the TD($\lambda$) error is often referred to as the generalized advantage estimate (GAE; Schulman et al., 2015).

The $\overline{\text{GPBE}}$ is defined using this TD($\lambda$) 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 one-step $\overline{\text{GPBE}}$ objective (Patterson et al., 2022). In the absence of function approximation, the $\overline{\text{GPBE}}$ and the $\overline{\text{GPBE}}(\lambda)$ objectives lead to the same solution, $v_\pi$, because their fixed points are both $v_\pi$. However, when function approximation is introduced, the choice of $\lambda$ strongly impacts the minimum-error solution. In practice, intermediate $\lambda$-values on the interval $(0, 1)$ will balance between solution quality, learning speed, and variance.

## 4 The Forward-View for Gradient TD($\lambda$) Methods

In this section, we develop several forward-view methods for optimizing the GPBE($\lambda$) under non-linear function approximation. Following the previous convention, we will overload the names GTD2($\lambda$) and TDC($\lambda$) introduced for the linear setting because we are strictly generalizing them to a broader function class.

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

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

We get a stochastic gradient descent update by sampling these expressions. For brevity throughout, let $V_t \stackrel{\text{def}}{=} \hat{v}(S_t, \boldsymbol{w}_t)$ and $H_t \stackrel{\text{def}}{=} \hat{h}(S_t, \boldsymbol{\theta}_t)$. 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($\lambda$) is a standard saddle-point update and should converge to a local optimum of the $\overline{\text{GPBE}}(\lambda)$ objective.

**TDC($\lambda$):** For TDC(0), we obtained an alternative gradient correction 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 $\delta_t$. 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($\lambda$) 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($\lambda$) 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($\lambda$) rather than GTD2($\lambda$), although, in our experiments, we find it is better empirically.

**TDRC($\lambda$):** 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 $\lambda$-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 significant variance may arise when using importance sampling, which degrades performance in paractice. In the next section we incorporate these forward-view updates into PPO, an algorithm that makes use of both replay and on-policy trajectories.

## 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 updates 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($\lambda$) error in Eq. (6). In practice, however, PPO updates must truncate the 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 $\lambda$-return using that truncated GAE as:

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

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 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 $\lambda$-return target is computed once from the collected trajectory and is kept fixed for all the training epochs on that trajectory. Although many PPO implementations heuristically clip this loss, we remove this component from our algorithm for simplicity.

We now introduce *Gradient PPO* which changes the critic update for PPO to allow for Gradient TD($\lambda$) 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($\lambda$) 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 $\delta^{\lambda}_t$. As a result, we cannot use a stale target as in Eq. (13). Instead, we need to recompute $\delta^{\lambda}_t$ 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 $\lambda$-return is the length of the sequence sampled from a minibatch, $\tau$, rather than the full trajectory length $T$. A similar approach to incorporate the $\lambda$-return with replay buffers was previously introduced (Daley & Amato, 2019). 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 $\tau$, $\nabla_{\boldsymbol{w}}\delta_t^\lambda$ and $\delta_t^\lambda$ 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 show in Appendix C.1 that using this parallel approach for computation results in Gradient PPO having the same Steps Per Second (SPS) cost as PPO; there is no drop in runtime from these additional calculations. We also provide a full algorithm for PPO and Gradient PPO in Appendix C.

**Algorithm 1** Estimating TDRC($\lambda$) 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 Investigation 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 $\lambda$. 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 single hyperparameter configuration that worked reasonably well across all environments and evaluated it for 30 more seeds—the *two-stage approach* (Patterson et al., 2024). We provide the ranges of values we swept over in Appendix C.1 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.

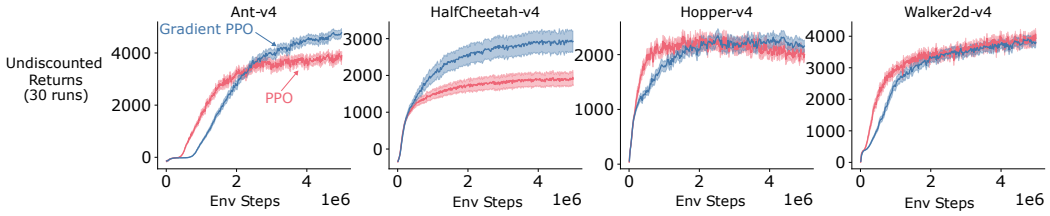


Figure 1: Gradient PPO and PPO evaluated on four MuJoCo environments. The solid lines are the mean performance averaged over 30 seeds, and the shaded area is the standard error.

We also investigated the utility of using TDRC($\lambda$) instead of TDC($\lambda$) and GTD2($\lambda$) 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.

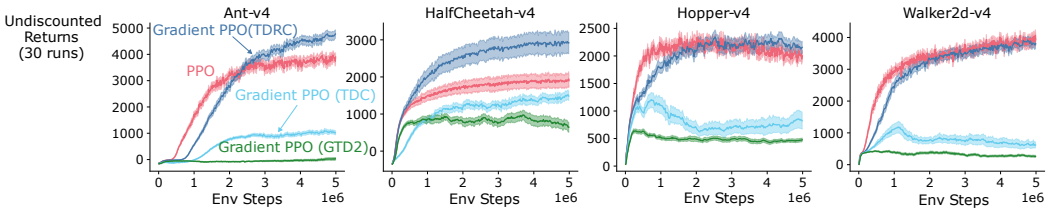


Figure 2: Gradient PPO variations evaluated on 4 MuJoCo environments. The solid lines are the mean performance averaged over 30 seeds, and the shaded area is the standard error.

## 6 The Backward View for Gradient TD($\lambda$) 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 $\overline{\text{GPBE}}(\lambda)$.

**GTD2**($\lambda$): As we prove below, the following backward-view updates are equivalent to the forward-view updates given in Eq. 8 and Eq. 9:

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

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

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.

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

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

**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($\lambda$) 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. □

**TDC($\lambda$):** 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($\lambda$) with off-policy corrections; an instantaneous correction that does not require eligibility traces; and GTD2($\lambda$)'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($\lambda$) 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($\lambda$):** 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.

Finally, we note that the backward view algorithms presented here do indeed update on every step, unlike PPO, but the proof above only shows equivalence at the end of the episode, like the original forward-backward equivalence of TD($\lambda$).

## 7 QRC($\lambda$): 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 use a truncated version of the updates, backward-view methods do not have this limitation.

### 7.1 QRC($\lambda$)

Extending the backward-view algorithms to action values is straightforward. Here, we present the extensions to Q($\lambda$), but similar extensions can be done to other action-value methods, such as SARSA($\lambda$). 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:

$$\delta_t = R_{t+1} + \gamma \max_{a' \in \mathcal{A}} \hat{q}(S_{t+1}, a', \boldsymbol{w}_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t)\,. \tag{26}$$

The gradient of the TD error becomes the following:

$$\nabla_{\boldsymbol{w}_t} \delta_t = \gamma \nabla_{\boldsymbol{w}_t} \left( \max_{a' \in \mathcal{A}} \hat{q}(S_{t+1}, a', \boldsymbol{w}_t) \right) - \nabla_{\boldsymbol{w}_t} \hat{q}(S_t, A_t, \boldsymbol{w}_t)\,. \tag{27}$$

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($\lambda$), which we refer to as QRC($\lambda$):

$$\begin{aligned}
\boldsymbol{z}_t^{\boldsymbol{w}} &= \gamma\lambda \boldsymbol{z}_{t-1}^{\boldsymbol{w}} + \nabla_{\boldsymbol{w}_t} \hat{q}(S_t, A_t, \boldsymbol{w}_t) \\
z_t^h &= \gamma\lambda z_{t-1}^h + H_t \\
\boldsymbol{z}_t^{\boldsymbol{\theta}} &= \gamma\lambda \boldsymbol{z}_{t-1}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} H_t \\
\Delta \boldsymbol{w}_t &= \delta_t \boldsymbol{z}_t^{\boldsymbol{w}} - H_t \nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t, \boldsymbol{w}_t) - z_t^h \nabla_{\boldsymbol{w}} \delta_t \\
\Delta \boldsymbol{\theta}_t &= \delta_t \boldsymbol{z}_t^{\boldsymbol{\theta}} - H_t \nabla_{\boldsymbol{\theta}} H_t - \beta \boldsymbol{\theta}_t
\end{aligned} \tag{28}$$

We can modify these updates to get QC($\lambda$), an update based on TDC($\lambda$), by simply setting $\beta = 0$. We can also get GQ($\lambda$), an update based on GTD2($\lambda$) by setting $\beta = 0$ and removing the gradient correction term (see Table 2). Finally, we follow Watkins' Q($\lambda$) 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 Investigation of QRC($\lambda$)

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

For Q($\lambda$) and QRC($\lambda$), we used SGD and performed a hyperparameter sweep for different values for the step size and $\lambda$. We tested each hyperparameter configuration in all environments and across 5 seeds. We then selected a single 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 D.1. For StreamQ, we used the hyperparameters suggested by the paper and the accompanying code, and we repeated the experiments for 30 seeds in all environments. Figure 3 shows the performance of all three algorithms across the 5 MinAtar environments, and in all environments, QRC($\lambda$) outperforms both StreamQ and Q($\lambda$).

We evaluated the other two gradient-based algorithms, QC($\lambda$) and GQ2($\lambda$). 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.

## 8 Conclusion

We proposed the $\overline{\text{GPBE}}(\lambda)$ objective, a multistep generalization of the Generalized Projected Bellman Error (Patterson et al., 2022) based on the $\lambda$-return. We derived three algorithms for optimizing

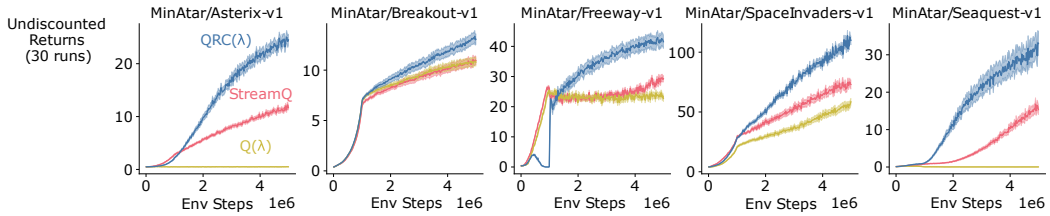


Figure 3: QRC($\lambda$), Q($\lambda$) and StreamQ algorithms evaluated on the five MinAtar environments. The solid lines are the mean performance averaged over 30 seeds, and the shaded regions are the corresponding standard errors.

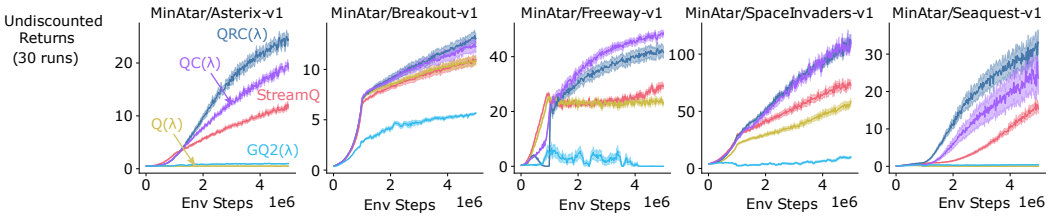


Figure 4: All gradient-based backward view algorithms evaluated on the 5 MinAtar environments. The solid lines are the mean performance averaged over 30 seeds, and the shaded regions are the corresponding standard errors.

the new objective both in the forward view and in the backward view. Of the three algorithms we developed, we showed that TDRC($\lambda$) is stable, fast, and results in 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. Further work remains to verify the convergence guarantees for TDC($\lambda$) and TDRC($\lambda$), and extending the gradient-based updates to more Deep RL algorithms.

## 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($\lambda$) 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

$$H_t \nabla_{\boldsymbol{w}} \hat{\delta}_t^{\lambda} = H_t \rho_t \nabla_{\boldsymbol{w}} \delta_t + H_t \gamma \lambda \rho_t \rho_{t+1} \nabla_{\boldsymbol{w}} \delta_{t+1} + H_t (\gamma\lambda)^2 \rho_t \rho_{t+1} \rho_{t+2} \nabla_{\boldsymbol{w}} \delta_{t+2} \ldots . \tag{29}$$

The total sum of these forward-view contributions is therefore

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

$$= (H_0 \rho_0) \nabla_{\boldsymbol{w}} \delta_0 + (H_0 \gamma \lambda \rho_0 \rho_1 + H_1 \rho_1) \nabla_{\boldsymbol{w}} \delta_1 + \dots \tag{30}$$

$$= z_0^h \nabla_{\boldsymbol{w}} \delta_0 + z_1^h \nabla_{\boldsymbol{w}} \delta_1 + \dots \tag{31}$$

$$= \sum_{t=0}^{\infty} z_t^h \nabla_{\boldsymbol{w}} \delta_t, \tag{32}$$

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

$$\sum_{t=0}^{\infty} \hat{\delta}_t^{\lambda} \nabla_{\boldsymbol{\theta}} H_t = \sum_{t=0}^{\infty} \delta_t z_t^{\boldsymbol{\theta}}, \tag{33}$$

and further note that

$$\hat{\delta}_t^{\lambda} \nabla_{\boldsymbol{\theta}} H_t = \rho_t \delta_t \nabla_{\boldsymbol{\theta}} H_t + \gamma \lambda \rho_t \rho_{t+1} \delta_{t+1} \nabla_{\boldsymbol{\theta}} H_t + (\gamma \lambda)^2 \rho_t \rho_{t+1} \rho_{t+2} \delta_{t+2} \nabla_{\boldsymbol{\theta}} H_t + \dots. \tag{34}$$

The total sum of these forward-view contributions is therefore

$$\sum_{t=0}^{\infty} \delta_t^{\lambda} \nabla_{\boldsymbol{\theta}} H_t = (\rho_0 \delta_0 \nabla_{\boldsymbol{\theta}} H_0 + \gamma \lambda \rho_0 \rho_1 \delta_1 \nabla_{\boldsymbol{\theta}} H_0 + \dots) + (\rho_1 \delta_1 \nabla_{\boldsymbol{\theta}} H_1 + \gamma \lambda \rho_1 \rho_2 \delta_2 \nabla_{\boldsymbol{\theta}} H_1 + \dots) + \dots \tag{35}$$

$$= \delta_0 (\rho_0 \nabla_{\boldsymbol{\theta}} H_0) + \delta_1 (\gamma \lambda \rho_0 \rho_1 \nabla_{\boldsymbol{\theta}} H_0 + \rho_1 \nabla_{\boldsymbol{\theta}} H_1) + \dots \tag{36}$$

$$= \delta_0 z_0^{\boldsymbol{\theta}} + \delta_1 z_1^{\boldsymbol{\theta}} + \dots \tag{37}$$

$$= \sum_{t=0}^{\infty} \delta_t z_t^{\boldsymbol{\theta}}, \tag{38}$$

which establishes Eq. (33) to prove Eq. (21) and complete the proof. □

**Acknowledgments**

We would like to thank NSERC, CIFAR, and Amii for research funding and the Digital Research Alliance of Canada for the computational resources. We would also like to thank Vlad Tkachuk for providing feedback on the paper draft.

## 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 $\lambda$-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($\lambda$): 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, Adam White, and Martha White. A generalized projected Bellman error for off-policy value estimation in reinforcement learning. *Journal of Machine Learning Research*, 2022.

Andrew Patterson, Victor Liao, and Martha White. Robust losses for learning value functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical design in reinforcement learning. *Journal of Machine Learning Research*, 25(318):1–63, 2024.

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 Sutton and Andrew Barto. *Reinforcement learning: An introduction*. 2018.

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 N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 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 ($\lambda$) 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 $\pi$. 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 $\hat{\delta}_t \stackrel{\text{def}}{=} \rho_t \delta_t$, 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($\lambda$) error is

$$\hat{\delta}_t^\lambda \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} (\gamma\lambda)^i \left( \prod_{j=0}^{i} \rho_{t+j} \right) \delta_{t+i} = \rho_t (\gamma\lambda \hat{\delta}_{t+1}^\lambda + \delta_t). \tag{39}$$

The backward view traces will then be defined as follows:

$$z_t^h \stackrel{\text{def}}{=} \rho_t (\gamma\lambda z_{t-1}^h + h_t)\,, \tag{40}$$

$$\boldsymbol{z}_t^{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \rho_t (\gamma\lambda \boldsymbol{z}_{t-1}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} h_t)\,, \tag{41}$$

## C Forward-View Algorithms

In this section, we provide the pseudocodes for the forward-view algorithms. We start with PPO in algorithm 2. PPO alternates between two main components: collecting a fixed-length trajectory of interactions using the current policy, and performing several steps of gradient updates using the collected trajectory. The gradient updates involve updating the value function towards an estimate of the $\lambda$-return based on the collected trajectory, and updating the policy parameters using the log-likelihood ratio. These steps are illustrated in algorithm 2. In the algorithm, we refer to the policy and value function parameters used during the collection of the trajectory as $\boldsymbol{\theta}_{\text{old}}$ and $\mathbf{w}_{\text{old}}$, respectively, while we refer to the most recent policy and value parameters as $\mathbf{w}_{\text{new}}$ and $\boldsymbol{\theta}_{\text{new}}$, those would be the result of the most recent mini-batch update.

Algorithm 3 shows the modifications needed to combine PPO with TDRC($\lambda$) to produce Gradient PPO. We highlight the main changes over the PPO algorithm in blue. Gradient PPO introduces three new parameters: 1) Truncation length, $T$, which represents the sequence length used to compute the $\lambda$-returns. 2) Learning rate for the auxiliary variable $h$, $\alpha_h$. 3) Regularization coefficient $\beta$, we found that simply setting $\beta = 1$ worked well for all the experiments we presented. Additionally, for the gradient updates, we construct a mini-batch of sequences and estimate the $\lambda$-returns per sequence. Note that a major change over PPO is that the $\lambda$-returns are estimated per minibatch using the latest parameters rather than stale estimates. As mentioned in the main paper, this change allows us to take the gradient of the $\delta^\lambda$ with respect to the latest parameters, which is needed for updates of the GTD algorithms.

### C.1 Experimental Details of PPO and Gradient PPO

For PPO, we used the default hyperparameters widely used for PPO, which reproduce the best-reported performance for PPO on MuJoco (Huang et al., 2022). We include those hyperparameters in Table 3 for completeness.

For Gradient PPO, we first performed a hyperparameter sweep for $\lambda$, actor learning, and critic learning rate. We show the ranges used for the sweep in Table 5. We repeated the experiments for each hyperparameter configuration in that sweep over 5 seeds. Based on that sweep, we chose the

**Algorithm 2** PPO Algorithm

Input: a differentiable policy parametrization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parametrization $\hat{v}(s, \mathbf{w})$
Algorithm parameters: learning rate $\alpha$, rollout length $\tau$, mini-batche size $n$, number of epochs $k$, value coefficient $c_1$, entropy coefficient $c_2$, clip coefficient $\epsilon$, max gradient norm $c$.
**for** iteration $= 1, 2, \cdots$ **do**
  Run $\pi_{\text{old}}(a|s, \boldsymbol{\theta}_{\text{old}})$ for $\tau$ steps. ▷ Collect a trajectory of interactions
  Calculate $\hat{v}(s_{t+\tau}, \mathbf{w}_{\text{old}})$ ▷ For bootstrapping
  Set $\hat{A}^{(\gamma,\lambda)}_{t+\tau} = 0$ ▷ initialization GAE estimate.
  **for** $j = t + \tau - 1, \ldots,$ t **do** ▷ Calculating GAE using the collected trajectory of interactions.
    $\delta_j = R_{j+1} + \gamma \hat{v}(s_{j+1}, \mathbf{w}_{\text{old}}) - \hat{v}(s_j, \mathbf{w}_{\text{old}})$
    $\hat{A}^{(\gamma,\lambda)}_j = \delta_j + \gamma\lambda \hat{A}^{(\gamma,\lambda)}_{j+1}$
    $\hat{G}^{\lambda}_j = \hat{A}^{(\gamma,\lambda)}_j + \hat{v}(s_j, \mathbf{w}_{\text{old}})$
  **end for**
  **for** epoch = 1, …, k **do** ▷ Learning
    Shuffle the transitions
    Divide the data into $m$ mini-batches of size $n$, where $m = \tau/n$.
    **for** mini-batch $= 1, \ldots, m$ **do**
      Calculate: $\log \pi_{new}(a|s, \boldsymbol{\theta}_{\text{new}})$, $\hat{v}(s, \mathbf{w}_{\text{new}})$ for samples in the mini-batch.
      Normalize $\hat{A}^{(\gamma,\lambda)}$ estimates per batch.
      **Policy objective:** $L_{\text{p}} = -\frac{1}{n}\sum_{j=1}^{n} \min(r_j \hat{A}^{(\gamma,\lambda)}_{j,\mathbf{w}_{\text{old}}}, \text{clip}_\epsilon(r_j)\hat{A}^{(\gamma,\lambda)}_{j,\mathbf{w}_{\text{old}}})$,
      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:** $L_{\text{v}} = \frac{1}{n}\sum_{j=1}^{n} \max((\hat{v}(s_j, \mathbf{w}_{\text{new}}) - \hat{G}^{\lambda}_{j,\mathbf{w}_{\text{old}}})^2, (\text{clip}_\epsilon(\hat{v}) - \hat{G}^{\lambda}_j)^2)$,
      where $\text{clip}_\epsilon(\hat{v}) = \text{clip}(\hat{v}(s_j, \mathbf{w}_{\text{new}}), 1-\epsilon, 1+\epsilon)$
      **Calculate the entropy of the policy:** $L_s = \frac{1}{n}\sum_{j=1}^{n} S(\pi(s_j, \boldsymbol{\theta}_{\text{new}}))$
      **Calculate the total loss:** $L = L_{\text{p}} + c_1 L_{\text{v}} - c_2 L_s$
      **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**

hyperparameters that generally performed well across all environments, Table 4. Finally, we fixed those hyperparameters configurations for all environments and ran the algorithm again for 30 seeds using those hyperparameters.

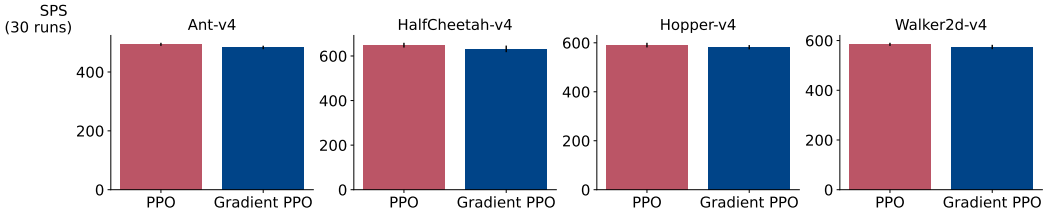


Figure 5: SPS for Gradient PPO and PPO evaluated on four MuJoCo environments. The bars indicate the mean across 30 runs and the black bars indicate the standard error.

**Algorithm 3** Gradient PPO: PPO with TDRC($\lambda$)

Input: a differentiable policy parametrization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parametrization $\hat{v}(s, \mathbf{w})$
Input: a differentiable auxiliary function parametrization $\hat{h}(s, \boldsymbol{\theta}_h)$
Algorithm parameters: learning rate $\alpha$, rollout length $\tau$, mini-batche size $n$, number of epochs $k$, entropy coefficient $c_2$, clip coefficient $\epsilon$, max gradient norm $c$, Truncation Length $T$, $h$ learning rate $\alpha_h$, regularization coefficient $\beta = 1$
**for** iteration $= 1, 2, \cdots$ **do**
  Run $\pi_{\text{old}}(a|s, \boldsymbol{\theta}_{\text{old}})$ for $\tau$ steps. ▷ Collect a trajectory of interactions
  Calculate $\hat{v}(s_{t+\tau}, \mathbf{w}_{\text{old}})$ ▷ For bootstrapping
  Construct a batch of $\frac{\tau}{T}$ sequences, where each sequence is:
$\langle s_i, a_{i+1}, R_{i+1}, \log \pi_{old}(a_{i+1}|s_i, \boldsymbol{\theta}_{\text{old}}), \hat{v}(s_i, \mathbf{w}_{\text{old}})\rangle, \ldots$
$\langle s_{i+T}, a_{i+T+1}, R_{i+T+1}, \log \pi_{old}(a_{i+T+1}|s_{i+T}, \boldsymbol{\theta}_{\text{old}}), \hat{v}(s_{i+T}, \mathbf{w}_{\text{old}})\rangle$
  **for** epoch = 1, …, k **do** ▷ Learning
    Shuffle the sequences
    Divide the data into $m$ mini-batches of size $n$, where $m = \tau/(n * T)$.
    **for** mini-batch $= 1, \ldots, m$ **do**
      **for** $j = t + T - 1, \ldots,$ t **do** ▷ This loop is parallelized over the sequences.
        $\delta_j = R_{j+1} + \gamma \hat{v}(s_{j+1}, \mathbf{w}_{\text{new}}) - \hat{v}(s_j, \mathbf{w}_{\text{new}})$
        $\nabla \delta_{j_{\mathbf{w}_{\text{new}}}} = R_{j+1} + \gamma \nabla \hat{v}(s_{j+1}, \mathbf{w}_{\text{new}}) - \nabla \hat{v}(s_j, \mathbf{w}_{\text{new}})$
        $\delta_j^\lambda = \delta_j + \gamma \lambda \delta_{j+1}^\lambda$
        $\nabla \delta_j^\lambda = \nabla \delta_j + \gamma \lambda \nabla \delta_{j+1}^\lambda$
      **end for**
      Calculate: $\log \pi_{new}(a|s, \boldsymbol{\theta}_{\text{new}})$ for samples in the mini-batch.
      **Policy objective:** $L_{\text{p}} = -\frac{1}{n} \sum_{j=1}^{n} \min(r_j \delta_j^\lambda, \text{clip}_\epsilon(r_j) \delta_j^\lambda)$
      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:** $L_S = \frac{1}{n} \sum_{j=1}^{n} S(\pi(s_j, \boldsymbol{\theta}_{\text{new}}))$
      **Calculate the total loss:** $L = L_{\text{p}} - c_2 L_S$
      **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**

Finally, to show that the additional calculations don't affect the run time of Gradient PPO, we plotted the Steps Per Second (SPS) for both PPO and Gradient PPO across all environments. Figure 5 shows that the SPS values are almost the same moving from PPO to Gradient PPO.

## D Backward-View Algorithms

In this section, we provide the pseudocode and the hyperparameters details for the backward-view algorithms. Algorithm 4 shows the pseudocode for QRC($\lambda$) where at each timestep, the agent samples an action from an $\epsilon$-greedy policy and takes one step in the environment to observe the subsequent

reward and next state. Then, based on that transition make an update its parameters using the traces it is carrying and also updates those traces.

**Algorithm 4** QRC($\lambda$) Algorithm

Input: a differentiable state-value function parametrization $\hat{q}_{\mathbf{w}}$
Input: a differentiable auxiliary function parametrization $\hat{h}_{\theta}$
Algorithm parameters: learning rate $\alpha_q$, $h$ learning rate $\alpha_h$, exploration parameter $\epsilon$.
Initialize $\boldsymbol{z}_t^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$
Initialize $\boldsymbol{z}_t^{\boldsymbol{w}} \leftarrow \mathbf{0}$
Initialize $z_t^h \leftarrow 0$
Observe initial state $S_0$
**for** iteration $t = 1, 2, \cdots$ **do**
  Sample an action $A_t \sim \pi$ . ▷ We use an $\epsilon$-greedy policy.
  Take action $A_t$, observe $R_{t+1}$ and $S_{t+1}$.
  Compute $\delta_t$ and $\nabla_{\boldsymbol{w}_t} \delta_t$ according to Eq. 26 and Eq. 27, respectively.
  Update the traces $\boldsymbol{z}_t^{\boldsymbol{\theta}}$, $\boldsymbol{z}_t^{\boldsymbol{w}}$, and $z_t^h$ according to Eq.28.
  Compute $\Delta \boldsymbol{w}_t$ and $\Delta \boldsymbol{\theta}_t$ according to Eq.28. ▷ To use the other algorithmic variants, replace those equations with other backward-view algorithms in Table 2.
  Update the parameters $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha_q \Delta \boldsymbol{w}_t$
  Update the parameters $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_h \Delta \boldsymbol{\theta}_t$
  **if** episode terminated or $A_t$ is non-greedy **then**
    reset the traces $\boldsymbol{z}_t^{\boldsymbol{\theta}}$, $\boldsymbol{z}_t^{\boldsymbol{w}}$, and $z_t^h$ to zeros.
  **end if**
**end for**

### D.1 Experimental Details of MinAtar

In our experiments with QRC($\lambda$), we used the normalization wrappers and the sparse initialization proposed by Elsayed et al. (2024). However, we used SGD as the optimizer presented Elsayed et al. (2024) can't be easily mapped to our updates. That choice was made so that the difference in performance between those algorithms can be associated with the Gradient TD updates.

For QRC($\lambda$), we first performed a hyperparameter sweep over $\lambda$, and the learning rate for both the value network and the $h$ network. We include the values used for the sweep in 6 and the final hyperaprameters used in Table 7.

Finally, we estimated the SPS for both QRC($\lambda$) and StreamQ, and we show the estimated values in Figure 6. We note that for StreamQ, we used the available implementations provided by the original paper to ensure reproducibility (Elsayed et al., 2024), which were implemented in PyTorch. However, for our QRC($\lambda$), we had an implementation in Jax, so we acknowledge the comparison of the SPS here has other confounding factors. Nevertheless, we show it to ensure that our runtime is not badly affected by the marginal additional computation required for the auxiliary variable $h$.

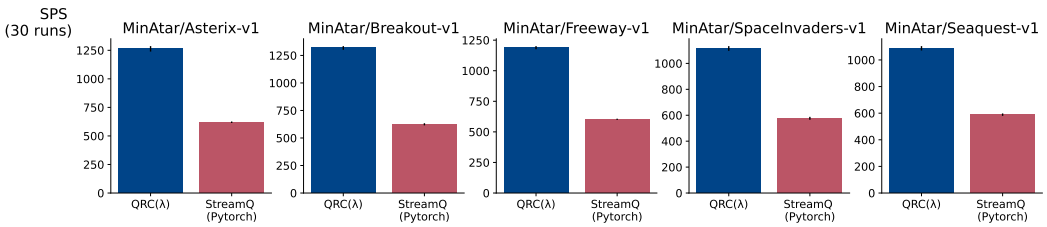


Figure 6: SPS for QRC($\lambda$) and StreamQ algorithms on MinAtar environments.

| 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 | 64 |
| GAE, $\lambda$ | 0.95 |
| Discount factor, $\gamma$ | 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 $\epsilon$ | $1 \times 10^{-5}$ |

Table 3: The default hyperparameters used for PPO. The hyperparmaeters values are based on the implementation details by Huang et al. (2022).

| 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) |
| $\lambda$ | 0.95 |
| Discount factor, $\gamma$ | 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, $\beta$ | 1.0 |
| Optimizer $\epsilon$ | $1 \times 10^{-5}$ |

Table 4: The hyperparameters used for Gradient PPO in all the MuJoCo experiments.

| Name | Sweep Range |
|---|---|
| $\lambda$ | $[0.7, 0.8, 0.9, 0.95]$ |
| 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, $\beta$ | $[1.0, 0.0]$ |

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

| Name | Default Value |
|---|---|
| $\lambda$ | [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, $\beta$ | [1.0,0.0] |

Table 6: Hyperparameters ranges used for the sweep experiments in MinAtar.

| Name | Default Value |
|---|---|
| $\lambda$ | 0.8 |
| Input Normalization | True |
| Optimizer | SGD |
| step size | 0.0001 |
| h step size | 0.1 |
| regularization coef, $\beta$ | 1.0/0.0 |
| start exploration $\epsilon$, | 1.0 |
| end exploration $\epsilon$, | 0.01 |
| exploration fraction | 0.2 |

Table 7: Final hyperparameters used for QRC($\lambda$) experiments with MinAtar