From Past to Future: Rethinking Eligibility Traces

Dhawal Gupta[1*], **Scott M. Jordan**[2], **Shreyas Chaudhari**[1], **Bo Liu**[3], **Philip S. Thomas**[1], **Bruno Castro da Silva**[1]

[1]University of Massachusetts,
[2]University of Alberta,
[3]Amazon
{dgupta, schaudhari, pthomas, bsilva}@cs.umass.edu, sjordan@ualberta.ca, boliucs@amazon.com

Abstract

In this paper, we introduce a fresh perspective on the challenges of credit assignment and policy evaluation. First, we delve into the nuances of eligibility traces and explore instances where their updates may result in unexpected credit assignment to preceding states. From this investigation emerges the concept of a novel value function, which we refer to as the *bidirectional value function*. Unlike traditional state value functions, bidirectional value functions account for both future expected returns (rewards anticipated from the current state onward) and past expected returns (cumulative rewards from the episode's start to the present). We derive principled update equations to learn this value function and, through experimentation, demonstrate its efficacy in enhancing the process of policy evaluation. In particular, our results indicate that the proposed learning approach can, in certain challenging contexts, perform policy evaluation more rapidly than TD(λ)—a method that learns forward value functions, v^π, *directly*. Overall, our findings present a new perspective on eligibility traces and potential advantages associated with the novel value function it inspires, especially for policy evaluation.

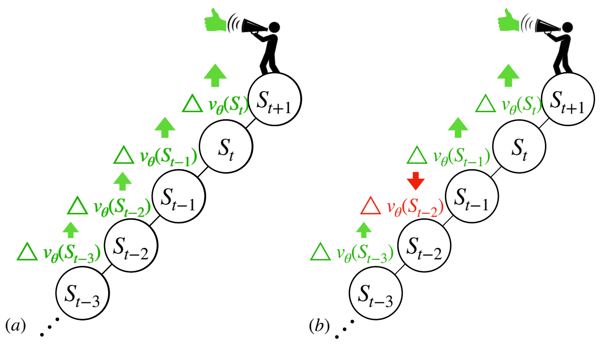
$\triangle\, v_\theta(S_t)$
S_{t+1}
S_t
$\triangle\, v_\theta(S_{t-1})$
S_{t-1}
$\triangle\, v_\theta(S_{t-2})$
S_{t-2}
$\triangle\, v_\theta(S_{t-3})$
S_{t-3}
(a)
(b)

Figure 1: Implementation of the backward view via TD(λ), adapted from Sutton and Barto (2018). Arrow sizes denote the magnitude of updates: (a) Expected update direction of past state values under the backward view, or TD(λ). (b) Potential misalignment in updates due to reliance on outdated gradient memory for past states (especially when using non-linear function approximation), which deviated from (a).

Introduction

Reinforcement Learning (RL) offers a robust framework for tackling complex sequential decision-making problems. The growing relevance of RL in diverse applications——from controlling nuclear reactors (Radaideh et al. 2021; Park et al. 2022) to guiding atmospheric balloons (Bellemare et al. 2020) and optimizing data centers (Li et al. 2019)—— underscores the need for more efficient solutions. Central to these solutions is addressing the *credit assignment problem*, which involves determining which actions most contributed to a particular outcome—a key challenge in sequential decision-making. The outcome of actions in RL may be delayed, posing a challenge in correctly attributing credit to the most relevant prior states and actions. This often requires a large number of samples or interactions with the environment. In real-world settings, this might be a constraint as system interactions are often risky or costly. Minimizing the sample and computational complexity of existing algorithms not only addresses these challenges but also facilitates the wider adoption of RL in various future applications.

[*]Corresponding author

Addressing the credit assignment problem given a temporal sequence of states and actions has been and continues to be an active area of research. A key concept in addressing this challenge is the Temporal Difference (TD) methods (Sutton 1984). In the context of TD methods, the *backward view* (Sutton and Barto 2018) offers an intuitive approach under which the agent aims to adjust the value of previous states to align with recent observations. As depicted in Figure 1(a), observing a positive outcome at a given time step, t, prompts the agent to increase the value estimates of all preceding states. This adjustment incorporates a level of discounting, accounting for the diminishing influence of distant preceding states on the observed outcome. This approach assumes that every prior state should share credit for the resultant outcome. One advantage of the backward view, discussed later, is that it can be efficiently implemented online and incrementally.

The TD(λ) method is a widely adopted algorithmic implementation of the backward view (Sutton 1984). Initially introduced for settings involving linear parameterizations of the value function, it has become standard in policy evaluation RL problems. However, as the complexity of RL problems increases with novel applications, there has been a shift towards adopting neural networks as function approximators, primarily due to their flexibility to represent many functions. Yet, this shift is not without challenges. In our work, we underscore one particular issue: when deploying TD(λ) with nonlinear function approximators, particular settings might cause it to update the value of previous states in ways inconsistent with the standard expectations regarding its behavior, that run counter to the core intuition underlying the backward view (Figure 1(**a**)). Figure 2 illustrates such a scenario, where after observing a *positive* outcome, the value of some prior states are *decreased* rather than increased. Therefore, the direction of these updates is contrary to the intended or expected ones.

In a later section, we delve deeper into why this issue with TD(λ) arises. Intuitively, the problem lies in how TD(λ) relies on an "eligibility trace vector": a short-term memory of the gradients of previous states' value functions. This trace vector is essentially a moving average of gradients of previously visited states—and it is used by TD(λ) to update the value of multiple states simultaneously. However, as the agent continuously updates state values online, such average gradients can become outdated.

In the policy evaluation setting with non-linear function approximation, gradient memory vector maintained by TD(λ) can become outdated, which can pose challenges, leading to state value updates that are misaligned with the intended behavior of the backward view. As a result, past states might receive updates that do not align with our intentions or expectations. Importantly, this issue does *not* occur under linear functions. This is because, in the linear setting, trace vectors equate to fixed-state feature vectors.

The contributions of this paper are as follows:

- We present a novel perspective under which eligibility traces may be investigated, highlighting specific scenarios that may lead to unexpected credit assignments to prior states.
- Stemming from our exploration of eligibility traces, we introduce *bidirectional value functions*. This novel type of value function captures both future and past expected returns, thus offering a broader perspective than traditional state value functions.
- We formulate principled update equations tailored for learning bidirectional value functions while emphasizing their applicability in practical scenarios.
- Through empirical analyses, we illustrate how effectively bidirectional value functions can be used in policy evaluation. Our results suggest that the methods proposed can outperform the traditional TD(λ) technique, especially in settings involving complex non-linear approximators.

Background & Motivation

Notation and Introduction to RL

Reinforcement learning is a framework for modeling sequential decision-making problems where an agent interacts with the environment and learns to improve its decisions based on its previous actions and the rewards it receives. The most common way to model such problems is as a Markov Decision Process (MDP), defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, d_0, \gamma)$, where $\mathcal{S}$ is the state space; $\mathcal{A}$ is the action space; $P(S_{t+1} = s' | S_t = s, A_t = a)$ is a transition function describing the probability of transitioning to state s' given that the agent was in state s and executed action a; $R(S_t, A_t)$ is a bounded reward function; $d_0(S_0)$ is the starting state distribution; and γ is the discount factor. For ease of discussion, we also consider the case where *state features* may be used, e.g., to perform value function approximation. In this case, we assume a domain-dependent function $x : \mathcal{S} \to \mathbb{R}^d$ mapping states to corresponding d-dimensional feature representations. The agent behaves according to a policy denoted by π. In particular, while interacting with the environment and observing its current state at time t, S_t, the agent stochastically selects an action A_t according to its policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, where $\Delta(\mathcal{A})$ is the probability simplex over the actions; i.e., $A_t \sim \pi(S_t)$. After executing the action, the agent observes a reward value $R(S_t, A_t)$ and transitions to the next state, S_{t+1}. The goal of an RL agent is to find the policy $\pi^\star$ that maximizes the expected discounted sum of the rewards generated by it:

$$\pi^\star \in \arg\max_{\pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t) \right].$$

Policy Evaluation

The search for $\pi^\star$ is often made easier by being able to predict the future returns of a given policy—this is known as the *policy evaluation* problem. Return at time t is defined as $G_t \coloneqq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots$, where $R_t \coloneqq R(S_t, A_t)$. We define the value function for a state s as the expected return observed by an agent starting from state s following policy π, i.e., $v^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$.

Estimating v^π (i.e., performing policy evaluation) is an important subroutine required to perform *policy improvement*. It is often done in settings where the value function is approximated as a parametric function with parameters θ, $v_\theta(s) \approx v^\pi(s)$, where the weights are updated through an iterative process, i.e., $\theta_{t+1} = \theta_t + \triangle\theta_t$. A common way of determining the update term, $\triangle\theta_t$, is by assuming an update rule of the form

$$\triangle\theta_t = \alpha(G_t - v_{\theta_t}(S_t)) \frac{\partial v_\theta(S_t)}{\partial \theta}\Big|_{\theta=\theta_t},$$

where α is a step-size parameter. The two simplest algorithms to learn the value function are the Monte Carlo (MC) algorithm and the Temporal Differences (TD) learning algorithm. In the MC algorithm, updates are as follows

$$\triangle\theta_t = \alpha(R_t + \gamma G_{t+1} - v_{\theta_t}(S_t)) \frac{\partial v_\theta(S_t)}{\partial \theta}\Big|_{\theta=\theta_t},$$

where determining G_{t+1} requires that the agent wait until the end of an episode. TD learning algorithms replace the G_{t+1} term with the agent's current approximation or estimate of the return at the next step; i.e.,

$$\triangle\theta_t = \alpha(R_t + \gamma v_{\theta_t}(S_{t+1}) - v_{\theta_t}(S_t)) \frac{\partial v_\theta(S_t)}{\partial \theta}\Big|_{\theta=\theta_t}, \quad (1)$$

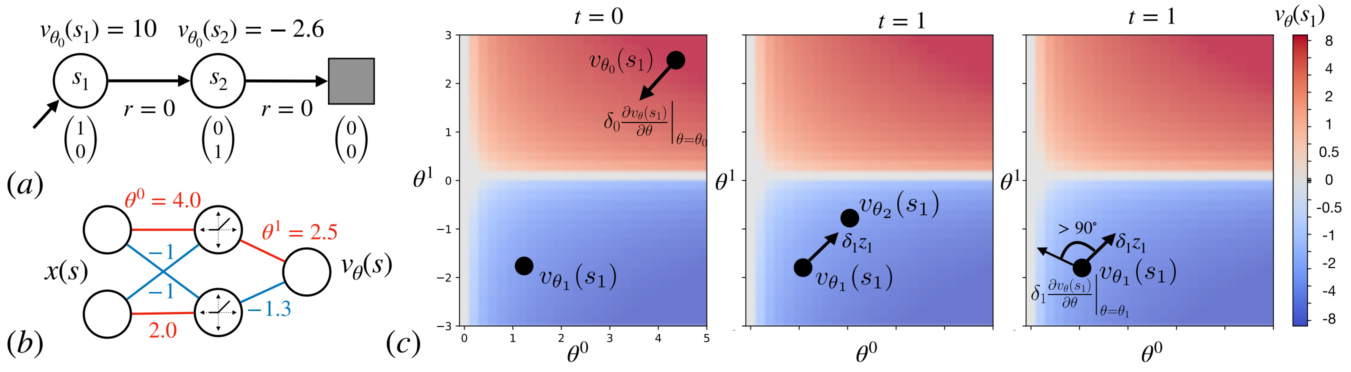

Figure 2: (a) A simple 2-state MDP. States are annotated with corresponding feature representations and values according to the approximator at time $t = 0$. (b) Functional form of the value function and respective parameters at $t = 0$. (c) On the (Left), a surface depicting the value of state s_1 for different parameter values (θ^0 and θ^1). After an update from $t = 0$ to $t = 1$, we can see (Middle) the update direction given by TD(λ) at $t = 1$ (i.e., $\delta_1 z_1$). On the (Right), we see that TD(λ)'s update direction is obtuse (and hence not aligned) to the direction based on the gradient of the *current* value function (i.e., $\delta_1 \frac{\partial v_\theta(s_1)}{\partial\theta}|_{\theta=\theta_1}$).

where $R_t + \gamma v_{\theta_t}(S_{t+1}) - v_{\theta_t}(S_t)$ is known as the TD error and denoted as δ_t. An important characteristic of the latter update is its online nature: the agent does not need to wait till the episode ends, since it does not rely on future variables. Thus, updates can be performed after each time step.

The family of λ-return algorithms is an intermediary between Monte Carlo (MC) and one-step TD(0) algorithms, using a smoothing parameter, λ.

TD(λ), which implements the backward view of the λ-return algorithm, relies only on historical information and can perform value function updates online. It accomplishes this by maintaining a trace vector e_t (known as the eligibility trace) encoding a short-term memory summarizing the history of the trajectory up to time t. This trace vector is then used to assign credit to the previous states visited by the agent based on the currently observed reward. In particular, the update term at time t is

$$\triangle\theta_t = \alpha\delta_t e_t, \text{ where } e_t := \sum_{i=0}^{t} (\lambda\gamma)^{t-i} \frac{\partial v_\theta(S_i)}{\partial\theta}\Big|_{\theta=\theta_i}. \quad (2)$$

Note that, when at state S_t, e_t can be computed recursively as the running average of the value function gradient evaluated at the states visited during the current episode:

$$e_t = \gamma\lambda e_{t-1} + \frac{\partial v_\theta(S_t)}{\partial\theta}\Big|_{\theta=\theta_t}, \quad (3)$$

where $z_{-1} = 0$; i.e., the trace is set to 0 at the start of episodes.

Note that eligibility traces, as defined above, can be used to perform credit assignment over a single episode. To perform credit assignment over multiple possible episodes, van Hasselt et al. (2020) introduced an *expected* variant of traces, $z(s)$, corresponding to a type of average trace over all possible trajectories ending at state s:

$$z(s) := \mathbb{E}[e_t | S_t = s]\,. \quad (4)$$

Misconception with Eligibility Traces

In TD(λ), the backward view uses the term $\frac{\partial v_\theta(S_i)}{\partial\theta}$ as a mechanism to determine how the value of a previously encountered state, S_i, will be updated. For instance, given an observed TD error at time t, δ_t, we may adjust the value of the state S_{t-3} by using the stored derivatives from time $t-3$, i.e., $\frac{\partial v_\theta(S_{t-3})}{\partial\theta}|_{\theta=\theta_{t-3}}$.Weight updates in TD($\lambda$) are implemented by maintaining a moving average of the gradients of the value functions related to previously-encountered states (see Eq. (2)). In particular, this weighted average determines how values of multiple past states will be updated given the currently observed TD error.

Notice, however, that this moving average aggregates information, say, about the gradient of the value of state S_{t-3} *assuming the value function at that time*, i.e., it aggregates information about the gradient $\frac{\partial v_\theta(S_{t-3})}{\partial\theta}|_{\theta=\theta_{t-3}}$. This gradient, however, represents the direction in which weights should be updated (based on the currently-observed outcome, δ_t) to update the *old, no-longer-in-use* value function, $v_{\theta_{t-3}}$, not the *current* value function, v_{θ_t}. The correct update direction based on the intuition in Figure 1**(a)**, by contrast, should be that given by the gradient of the *current* value function: $\frac{\partial v_\theta(S_{t-3})}{\partial\theta}|_{\theta=\theta_t}$. This indicates that the direction chosen by TD(λ) to update the value of previously encountered states may not align with the correct direction according to the value function's current parameters, θ_t, due to the use of **outdated gradients**. Mathematically, this discrepancy can be represented as (for $i > 0$):

$$\left(\frac{\partial v(S_{t-i})}{\partial\theta}\Big|_{\theta=\theta_{t-i}}\right)^\top \left(\frac{\partial v(S_{t-i})}{\partial\theta}\Big|_{\theta=\theta_t}\right) < 0.$$

Figure 2 depicts a simple example to highlight this problem—i.e., the fact that TD(λ) uses outdated gradients when performing updates. In this figure, we can see that the effective update direction of TD(λ) points in the direction opposite to the intended/correct one. We discuss the learning performance of both types of updates in Appendix A.

Conceptually, TD(λ) computes traces by combining previous derivatives of the value function to adjust the value of the state at time i, based on the TD error at time t, for $i < t$. Notably, this misconception was not present when TD(λ) was introduced (Sutton 1984). In its original form, the update was tailored for linear functions. In this case, the update is a function of the feature representation of each encountered state S_i, since $\frac{\partial v_\theta(S_i)}{\partial \theta} = x(S_i)$. Then, the trace is a moving average of features observed in previous steps; update issues are averted since the feature vector of any given state remains the same, independently of changes to the value function.

Recall that equation (2) corresponds to the TD(λ) update and that it highlights how it uses outdated gradients. A minor adjustment to this equation provides the desired update:

$$\triangle\theta_t = \alpha\delta_t\tilde{e}_t, \text{ where } \tilde{e}_t = \sum_{i=0}^{t}(\lambda\gamma)^{t-i}\frac{\partial v_\theta(S_i)}{\partial\theta}\Big|_{\theta=\theta_t}. \quad (5)$$

The key distinction is in using the latest/current weights, θ_t, during the trace calculation for prior states. For linear function approximations, both (2) and (5) produce identical updates.

A key advantage of the original update (2) is that it can be implemented online (in particular, via Eq. (3)). This requires only a constant computational and memory cost per update step. In contrast, if we were to use Eq. (5) to perform online updates, it would require computing the derivative for every state encountered up to time t. This makes the complexity of each update directly proportional to the episode's duration thus far. This computation becomes impractical for longer episodes. Furthermore, this approach negates the principle of incremental updates, as the computational cost per step increases based on episode length.

Let us adapt the expected trace formulation (4) to our Eq. (5). We start with an expected trace vector $\tilde{z}(s) := \mathbb{E}[\tilde{e}_t|S_t = s]$.[1] By substituting the value of $\tilde{e}_t$ into this expression and simplifying it, we obtain:

$$\begin{aligned}\mathbb{E}[\tilde{e}_t|S_t = s] &= \mathbb{E}\left[\sum_{i=0}^{t}(\lambda\gamma)^{t-i}\frac{\partial v_\theta(S_i)}{\partial\theta}\Big|_{\theta=\theta_t}|S_t = s\right]\\ &\underset{(a)}{=} \frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{i=0}^{t}\left((\lambda\gamma)^{t-i}v_\theta(S_i)\right)\Big|S_t = s\right]\Big|_{\theta=\theta_t}\\ &\underset{(b)}{=} \frac{\partial f(\theta, s)}{\partial\theta}\Big|_{\theta=\theta_t}. \end{aligned}\quad (6)$$

where $f(\theta, s) := \mathbb{E}\Big[\sum_{i=0}^{t}\left((\lambda\gamma)^{t-i}v_\theta(S_i)\right)\Big|S_t = s\Big]$. Step (a) uses linearity of expectation and gradients, and in (b) we define the inner term as a f. Notice that the resulting trace is the gradient of a function defined as the weighted sum of value approximations over different time steps.

Methodology

In the previous section, we showed that the expected trace update, when applied to Eq. (5), is the gradient of a function composed of the expected discounted sum of the value of states as approximated by v_θ. Let us consider what Eq. (6) would be at the point of convergence, $\theta^\star$, such that $v_{\theta^\star} = v^\pi$. At convergence, $f(\theta^\star, s)$ is

$$\begin{aligned}f(\theta^\star, s) &= \mathbb{E}\left[\sum_{i=0}^{t}\left((\lambda\gamma)^{t-i}v_{\theta^\star}(S_i)\right)\Big|S_t = s\right]\\ &= \mathbb{E}\left[\sum_{i=0}^{t}\left((\lambda\gamma)^{t-i}v^\pi(S_i)\right)\Big|S_t = s\right].\end{aligned}$$

Expanding the definition of f at $\theta^\star$ leads us to Lemma 1. Prior to delving into the lemma, let us define another quantity, $\overleftarrow{G}_t := \sum_{i=1}^{t}(\lambda\gamma)^i R_{t-i}$. This represents the discounted sum of rewards collected up to the current time step, where discounting is in the reverse direction with a factor of $\lambda\gamma$.

Lemma 1. *The discounted sum of the value function at the expected sequence of states in trajectories reaching state s at time t is*

$$\mathbb{E}\left[\sum_{i=0}^{t}(\lambda\gamma)^{t-i}v^\pi(S_i)\Big|S_t = s\right] = \frac{1}{1-\gamma^2\lambda}\mathbb{E}\Big[G_t + \overleftarrow{G}_t - \gamma(\lambda\gamma)^{t+1}G_0|S_t = s\Big].$$

Proof. Appendix C. □

In the equation above, the first two terms are the future and past discounted returns, as defined earlier. The third term, G_0, represents the return of the entire trajectory conditioned on the agent visiting state s at time t; it decays by a factor proportional to $\lambda\gamma$ as the episode progresses.

Similar to how v^π is defined as the expectation of G_t, we can define another type of value function (akin to v^π) representing the expected return observed by the agent up to the current time:

$$\overleftarrow{v}^\pi(s) := \mathbb{E}_\pi\left[\sum_{i=1}^{t}(\lambda\gamma)^i R_{t-i}|S_t = s\right]. \quad (7)$$

We call this value function, $\overleftarrow{v}$, the *backward value function*, with the arrow being used to indicate the temporal direction of the prediction.[2] This value represents the discounted sum of rewards the agent has received until now, where rewards earlier in the trajectory are more heavily discounted. The discount factor for this value function is $\lambda\gamma$, as opposed to the standard discount function, γ, used in the forward value function, $\overrightarrow{v}$. Henceforth, we use $\overrightarrow{v}$, v^π, and v interchangeably.

Let us now define another value function: the sum of the backward and forward value functions. This function combines the expected return to go and the return to get to a state s. Simply put, it is the summation of $\overleftarrow{v}$ and $\overrightarrow{v}$ at a given state:

$$\begin{aligned}\overleftrightarrow{v}(s_t) &:= \overleftarrow{v}(s_t) + \overrightarrow{v}(s_t)\\ &= \mathbb{E}\left[(\sum_{i=1}^{t}(\gamma\lambda)^i R_{t-i} + \sum_{i=0}^{\infty}\gamma^i R_{t+i})|S_t = s_t\right].\end{aligned}$$

[1]Note that van Hasselt et al. (2020) do not distinguish between e_t and $\tilde{e}_t$. This difference is often overlooked with traces, and one contribution of our work is to point out this difference.

[2]For brevity, we often drop π from value functions, using v_θ to represent the corresponding approximations of these functions.

We refer to this value function as the *bi-directional value function* and denote it as $\overleftrightarrow{v}$ to indicate the directions in which it predicts returns. Notice that we dropped the $\mathbb{E}\big[\gamma(\lambda\gamma)^{t+1}G_0|S_t = s\big]$ term when defining this value function because in the limit of $t \to \infty$, the influence of the starting state decreases, since $\lim_{t\to\infty}(\lambda\gamma)^{t+1} = 0$.

Notice that $\overleftrightarrow{v}$ represents (up to a constant scaling factor) the total discounted return received by the agent *throughout an entire trajectory* and passing through a specific state.

In this work, we wish to further investigate the properties of the value functions $\overleftrightarrow{v}$ and $\overleftarrow{v}$, and whether learning $\overleftrightarrow{v}$ and $\overleftarrow{v}$ can facilitate the identification of the forward value function, v^π. More precisely, in the next sections we:

- Investigate formal properties of these value functions in terms of learnability and convergence;
- Design principled and practical learning rules based on online stochastic sampling;
- Evaluate whether learning $\overleftrightarrow{v}^\pi$ may result in a more efficient policy evaluation process, particularly in settings where standard methods may struggle.

Bellman Equations and Theory

In this section, we show that the two newly introduced value functions ($\overleftrightarrow{v}$ and $\overleftarrow{v}$) have corresponding variants of Bellman equations. Previous work (Zhang, Veeriah, and Whiteson 2020) has shown that $\overleftarrow{v}$ allows for a recursive form (i.e., a Bellman equation), but our work is the first to present a Bellman equation for $\overleftrightarrow{v}$. We also prove that these Bellman equations, when used to define corresponding Bellman operators, are contraction mappings; thus, by applying such Bellman updates, we are guaranteed to converge to the corresponding value functions in the tabular setting.

First, we present the standard Bellman equation for the forward value function, $\overrightarrow{v}$:

$$\overrightarrow{v}^\pi(s_t) = r^\pi(s_t) + \gamma \sum_{s_{t+1}} \overrightarrow{\mathcal{P}^\pi}(s_{t+1}|s_t)\overrightarrow{v}^\pi(s_{t+1}),$$

wherein we define $r^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)R(s,a)$ and $\overrightarrow{\mathcal{P}^\pi}(s'|s) = \sum_{a\in\mathcal{A}} \pi(a|s)P(s'|s,a)$. This is the standard Bellman equation for the forward value function. Similarly, we can show that the Bellman equation for the *backward* value function can be written as

$$\overleftarrow{v}^\pi(s_t) = \lambda\gamma\overleftarrow{r^\pi}(s_t) + \lambda\gamma \sum_{s_{t-1}} \overleftarrow{\mathcal{P}^\pi}(s_{t-1}|s_t)\overleftarrow{v}(s_{t-1}).$$

The expression above can be proved by applying the recursive definition of $\overleftarrow{v}$ on (7), where $\overleftarrow{\mathcal{P}^\pi}(s_{t-1}|s_t)$ and $\overleftarrow{r^\pi}(s_t)$ are the backward-looking transition and reward functions. Appendix B presents further details about these definitions, and Appendix D shows the proof/complete derivation of the above Bellman equation.

Theorem 2. *Given the Bellman equations for $\overrightarrow{v}^\pi$ and $\overleftarrow{v}^\pi$, the Bellman equation for $\overleftrightarrow{v}^\pi$ is*

$$\begin{aligned}\overleftrightarrow{v}^\pi(s_t) ={}& \tfrac{1}{1+\gamma^2\lambda}\Big(r^\pi(s_t)(1-\gamma^2\lambda)+\\ &\gamma\sum_{s_{t+1}}\overrightarrow{\mathcal{P}^\pi}(s_{t+1}|s_t)\overleftrightarrow{v}^\pi(s_{t+1})+\\ &\lambda\gamma\sum_{s_{t-1}}\overleftarrow{\mathcal{P}^\pi}(s_{t-1}|s_t)\overleftrightarrow{v}^\pi(s_{t-1})\Big).\end{aligned}$$

Proof. We provide a proof sketch here. The complete proof is in Appendix E. To prove this result, we first recall that, by definition, $\overleftrightarrow{v}$ is the sum of $\overleftarrow{v}$ and $\overrightarrow{v}$. We the expand these terms into their corresponding Bellman equations. Further simplification of terms leads to the above Bellman equation. □

An important point to note in the above equation is that the value of $\overleftrightarrow{v}(s_t)$ bootstraps from the value of the previous state ($\overleftrightarrow{v}(s_{t-1})$), as well as the value of the next state ($\overleftrightarrow{v}(s_{t+1})$), which makes sense since this value function does look in the past as well as future. Another observation regarding this equation is the division by a factor of $\frac{1}{1+\lambda\gamma^2}$, which can be viewed as a way to normalize the effect of summing overlapping returns from two bootstrapped state values.

Using the Bellman equations for $\overleftarrow{v}_i$ and $\overleftrightarrow{v}_i$, we now define their corresponding Bellman operators, $\overleftarrow{\mathcal{T}}$ and $\overleftrightarrow{\mathcal{T}}$, as:

$$\overleftarrow{\mathcal{T}}(\overleftarrow{v}_i(s)) := \lambda\gamma\overleftarrow{r^\pi}(s) + \lambda\gamma\sum_{s'}\overleftarrow{\mathcal{P}^\pi}(s'|s)\overleftarrow{v}_i(s'), \quad (8)$$

and

$$\begin{aligned}\overleftrightarrow{\mathcal{T}}(\overleftrightarrow{v}_i(s')) :={}& \tfrac{1}{1+\gamma^2\lambda}(r^\pi(s')(1-\gamma^2\lambda)+ \quad (9)\\ &\gamma\sum_{s''}\overrightarrow{\mathcal{P}^\pi}(s''|s')\overleftrightarrow{v}_i(s'')+\\ &\lambda\gamma\sum_{s}\overleftarrow{\mathcal{P}^\pi}(s|s')\overleftrightarrow{v}_i(s)).\end{aligned}$$

Assumption 1. *The Markov chain induced by π is ergodic.*

Theorem 3. *Under Assumption 1, the limit $\lim_{t\to\infty}\mathbb{E}\Big[\overleftarrow{G}_t|S_t = s\Big]$ exists and the operator defined in* (8) *is a contraction mapping; and hence, repeatedly applying it leads to convergence to $\overleftarrow{v}^\pi$.*

Proof. The proof is provided in the Appendix F. □

Theorem 4. *Under Assumption 1, the limit $\lim_{t\to\infty}\mathbb{E}\Big[\overleftarrow{G}_t + G_t|S_t = s\Big]$ exists and the operator defined in* (9) *is a contraction mapping; and hence, repeatedly applying it leads to convergence to $\overleftrightarrow{v}^\pi$.*

Proof. The proof is provided in the Appendix F. □

Online and Incremental Update Equations

In the previous section, we introduced the Bellman operators for two value functions and proved that their corresponding operators are contractions. We would now like to derive an update equation allowing agents to learn such value functions from stochastic samples. Similar to how the TD(0) update rule is motivated by its corresponding Bellman operator, we can define similar update rules for $\overleftrightarrow{v}$ and $\overleftarrow{v}$. Let us parameterize our value functions as follows: $\overrightarrow{v}_\theta, \overleftarrow{v}_\phi, \overleftrightarrow{v}_\psi$. The TD(0) equivalent of the update equations for the parameters of $\overleftarrow{v}_\phi$ and $\overleftrightarrow{v}_\psi$ value functions are presented below.

Update for ψ and ϕ :

$$\triangle\psi_t = \alpha \overleftrightarrow{\delta}_t \frac{\partial \overleftrightarrow{v}_\psi(S_t)}{\partial \psi}\Big|_{\psi=\psi_t}, \quad \triangle\phi_t = \alpha \overleftarrow{\delta}_t \frac{\partial \overleftarrow{v}_\phi(S_t)}{\partial \phi}\Big|_{\phi=\phi_t}. \tag{10}$$

where the corresponding TD error are defined as $\overleftrightarrow{\delta}_t := \frac{1}{1+\gamma^2\lambda}(R_t(1-\gamma^2\lambda) + \gamma\overleftrightarrow{v}_{\psi_t}(S_{t+1}) + \gamma\lambda\overleftrightarrow{v}_{\psi_t}(S_{t-1})) - \overleftrightarrow{v}_{\psi_t}(S_t))$ and $\overleftarrow{\delta}_t := \lambda\gamma R_{t-1} + \lambda\gamma\overleftarrow{v}_{\phi_t}(S_{t-1}) - \overleftarrow{v}_{\phi_t}(S_t)$.

The above update equations (as is the case with standard TD methods) can be computed online and incrementally; i.e., they incur a fixed computation cost per step and thus allow us to distribute compute evenly throughout an agent's trajectory.

Note that when implementing such updates, we can use a scalar value to store the backward return and obtain an online version of the Monte Carlo update for $\overleftarrow{v}$, i.e., $\overleftarrow{G}_t = \lambda\gamma\overleftarrow{G}_{t-1} + \lambda\gamma R_{t-1}$, leading to the following online update:

$$\triangle\phi_t = \alpha(\overleftarrow{G}_t - \overleftarrow{v}_{\phi_t}(S_t))\frac{\partial \overleftarrow{v}_\phi(S_t)}{\partial \phi}\Big|_{\phi=\phi_t}. \tag{11}$$

where we define $\overleftarrow{G}_0 = 0$ and start updating $\overleftarrow{v}$ at time step $t = 1$. In Appendix G, we present several update equation variants that rely on other types of value functions.

Experiments

We investigate three questions: **RQ1**: Can we jointly parameterize all three value functions such that learning each of them individually helps to learn the other two? **RQ2**: Can learning such value functions facilitate/accelerate the process of evaluating forward value functions compared to standard techniques like TD(λ)? **RQ3**: What is the influence of λ on the method's performance?

Parameterization (RQ1)

We want to identify a parameterization for our value functions such that training $\overleftrightarrow{v}$ helps learn $\overrightarrow{v}$, i.e., the value function we care about. We can leverage the mathematical property that $\overleftrightarrow{v} = \overleftarrow{v} + \overrightarrow{v}$ to learn $\overrightarrow{v}$ such that these two functions are interdependent and allow for the other to be inferred.

Figure 3 shows a possible way to parameterize the value functions. In this case, we parameterize $\overleftrightarrow{v}$ as the sum of the other two heads of a single-layer neural network. In particular, $\overleftrightarrow{v} = \overleftarrow{v} + v$, and so $\theta = \{w^1, w^2\}$, $\phi = \{w^1, w^3\}$, and $\psi = \{w^1, w^2, w^3\}$. We refer to this parameterization as **BiTD-FR**. We can similarly fully parameterize the forward value function with all the weights as shown in Appendix H (Figure 7b), wherein $v = \overleftrightarrow{v} - \overleftarrow{v}$. In this case, the parameterization becomes $\theta = \{w^1, w^2, w^3\}$, $\phi = \{w^1, w^3\}$, and $\psi = \{w^1, w^2\}$; i.e., we make use of all the weights to parameterize v. We refer to this variant as **BiTD-BiR**. Similarly, for completeness, we define a third parameterization as $\overleftarrow{v} = \overleftrightarrow{v} - v$, wherein $\theta = \{w^1, w^2\}$, $\phi = \{w^1, w^2, w^3\}$, and $\psi = \{w^1, w^3\}$. We call this parameterization **BiTD-FBi**. Note that we have only shown a 1-layer neural network, but

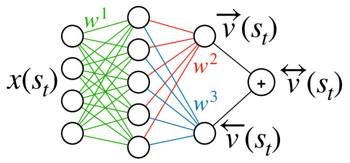

Figure 3: Parameterizing the three value functions: We parameterize $\overleftrightarrow{v}$ to be sum of the other two value functions.

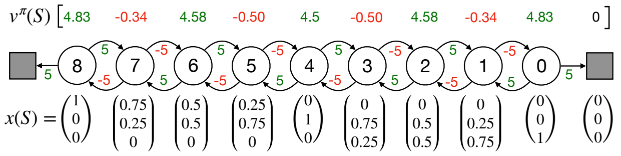
$v^\pi(S)$ [4.83 -0.34 4.58 -0.50 4.5 -0.50 4.58 -0.34 4.83 0]
8 7 6 5 4 3 2 1 0
5 -5
$x(S) =$

Figure 4: Chain Domain

w^1 can be replaced with any deep neural network and the parameterization would remain valid. Our formulation simply imposes more structure in our value function representation.

Training this network is also straightforward since we can estimate all three value functions with a single forward pass. We can also compute the losses of all three heads with their respective TD/MC updates as shown in (1), (10) and (11).

Policy Evaluation (RQ2 & RQ3)

We study the utility of using these value functions in a standard prediction task. One issue that might arise in value function approximation occurs when trying to approximate non-smooth value functions—i.e., value functions that might change abruptly w.r.t. to the input feature. In RL, this implies that the value of spatially similar states may differ vastly.

For our prediction problem, we consider a chain domain with 9 states (Sutton and Barto 2018) as depicted in Figure 4. The initial state is drawn from a uniform distribution over the state space. The agent can only take two actions (go left and go right) and the ending states of the chain are terminal. We use a feature representation (motivated by Boyan's chain (Boyan 2002)) such that the values of nearby states are forced to generalize over multiple features—a property commonly observed in continuous-state RL problems. To simulate a highly irregular value function, we define a reward function that fluctuates between -5 and +5 between consecutive states.

We evaluate each TD learning algorithm (along with the Monte Carlo variant for learning $\overleftarrow{v}$) in terms of their ability to approximate the value function of a uniform random policy, $\pi(\texttt{left}|\cdot) = \pi(\texttt{right}|\cdot) = 0.5$, under a discount factor of $\gamma = 0.99$. The analytically derived value function is shown in Figure 4 alongside the feature representation used for each state. In our experiments, we sweep over multiple values of learning rate (α) and λ. We use the learning rate for all value function heads. Each run corresponds to 50K training/environment steps, and we average the loss function over 100 seeds. We used a single-layer neural network with 9 units in the hidden layer and ReLU as the non-linearity.

Figure 5 depicts the results of this experiment with hyperparameters optimized for Area Under Curve (AUC). **(RQ2)**

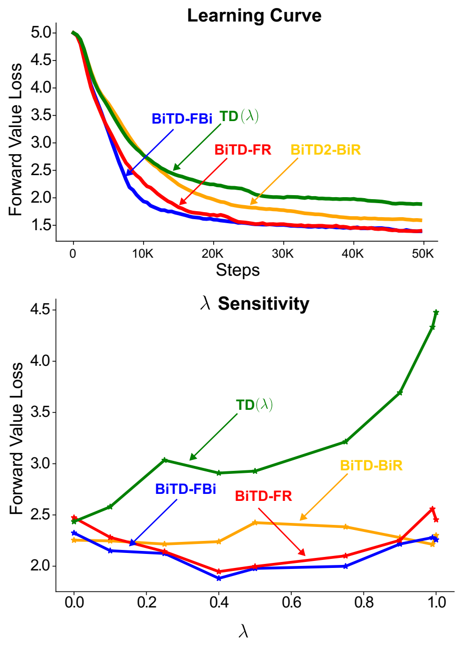

Figure 5: Experimental results for prediction in random chain domain. The y axis shows the MSTDE error of the forward value function. (top) Best performing parameter setting for BiTD-FR, BiTD-BiR, BiTD-FBi and standard TD(λ). (bottom) We compare all BiTD variants and TD(λ) for different values of λ; notice that any values $\lambda > 0$ are detrimental to TD(λ)'s performance, but can aid in performing policy evaluation using the proposed framework.

From Fig. 5**(top)** we can see how all variants of **BiTD** achieve a lower MSTDE loss than TD(λ) for a given number of samples. **(RQ3)** To investigate the role of λ in the efficiency of policy evaluation, we analyze Fig. 5**(bottom)**. From this figure, we can see that TD(λ)'s performance deteriorates strictly as the value of λ increases and that it performs best for $\lambda = 0$. We also notice that **BiTD-FR** performs similarly to TD for $\lambda = 0$, but its performance is better for intermediate values of λ (with the best performance being for $\lambda = 0.4$). Furthermore, notice that among different BiTD methods, the ones that directly approximate v (BiTD-FR and BiTD-FBi) seem to perform better than the ones that indirectly approximate it, like BiTD-BiR. Appendix H provides detailed plots with standard error bars (as well as the sensitivity of methods w.r.t α and λ) to better understand how α may affect methods.

Literature Review

The successful application of eligibility traces has been historically associated with linear function approximation (Sutton and Barto 2018). The update rules for eligibility traces, explicitly designed for linear value functions, encounter ambiguities when extended to nonlinear settings. The fact that the RL community started to use non-linear function approximations more often (due to the rise of deep RL) led to the wider use of experience replay, making eligibility traces hard to properly deploy. Nonetheless, several works have tried to adapt traces for deep RL (Tesauro 1992; Elfwing, Uchibe, and Doya 2018). Traces have found some utility in methods such as advantage estimation (Schulman et al. 2017). One interesting interpretation, proposed by van Hasselt et al. (2020), applies expected traces to the penultimate layer in neural nets while maintaining the running trace for the remaining network. Traces have also been modified to be combined with experience replay (Daley and Amato 2018).

Backward TD learning offers a new perspective to RL by integrating "hindsight" into the credit assignment process. Unlike traditional forward-view TD learning, which defines updates based on expected future rewards from present decisions, backward TD works retrospectively, estimating present values from future outcomes. This shift to a "backward view" has spurred significant advancements. Chelu, Precup, and van Hasselt (2020) underscored the pivotal roles of both foresight and hindsight in RL, illustrating their combined efficacy in algorithmic enhancement. Wang et al. (2021) leveraged backward TD for offline RL, demonstrating its potential in settings where data collection proves challenging. Further, the efficacy of backward TD learning methods in imitation learning tasks was highlighted by Park and Wong (2022). Zhang, Veeriah, and Whiteson (2020) reinforced the need for retrospective knowledge in RL, underscoring the significance of the backward TD methods.

One may consider learning $\overleftrightarrow{v}$ and $\overleftarrow{v}$ as auxiliary tasks. Unlike standard learning scenarios where auxiliary tasks may not directly align with the value function prediction objective, in our case, learning $\overleftrightarrow{v}$ and $\overleftarrow{v}$ results in auxiliary tasks that *directly* complement and align with the primary goal of learning the forward value function. Auxiliary tasks, when incorporated into RL, often act as catalysts, refining the primary task's learning dynamics. Such tasks span various functionalities—next-state prediction, reward forecasting, representation learning, and policy refinement, to name a few (Lin et al. 2019; Rafiee et al. 2022). The "Unsupervised Auxiliary Tasks" framework, introduced by Jaderberg et al. (2017) demonstrates how auxiliary tasks can enhance feature representations, benefiting primary and auxiliary tasks.

Conclusion and Future Work

In this work, we unveiled an inconsistency resulting from the combination of eligibility traces and non-linear value function approximators. Through a deeper investigation, we derived a new type of value function—a *bidirectional value function*—and showed principled update rules and convergence guarantees. We also introduced online update equations for stochastic sample-based learning methods. Empirical results suggest that this new value function might surpass traditional eligibility traces in specific settings, for various values of λ, offering a novel perspective to policy evaluation.

Future directions include, e.g., extending our on-policy algorithms to off-policy prediction. Another promising direction relates to exploring analogous functions but in the context of control rather than prediction. We would also like to investigate the hypothesis that bidirectional value functions (akin to average reward policy evaluation methods, which model complete trajectories) may be a first step towards unifying discounted and average-reward RL settings.

Acknowledgments

We thank Yash Chandak for his valuable feedback. We are also immensely grateful to the three anonymous reviewers and the meta-reviewer for their insights and feedback.

References

Bellemare, M. G.; Candido, S.; Castro, P. S.; Gong, J.; Machado, M. C.; Moitra, S.; Ponda, S. S.; and Wang, Z. 2020. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836): 77–82.

Boyan, J. A. 2002. Technical Update: Least-Squares Temporal Difference Learning. *Mach. Learn.*, 49(2-3): 233–246.

Chelu, V.; Precup, D.; and van Hasselt, H. P. 2020. Forethought and Hindsight in Credit Assignment. In *Advances in Neural Information Processing Systems*.

Daley, B.; and Amato, C. 2018. Efficient Eligibility Traces for Deep Reinforcement Learning. *CoRR*, abs/1810.09967.

Elfwing, S.; Uchibe, E.; and Doya, K. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2017. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *5th International Conference on Learning Representations, ICLR 2017*.

Li, Y.; Wen, Y.; Tao, D.; and Guan, K. 2019. Transforming cooling optimization for green data center via deep reinforcement learning. *IEEE transactions on cybernetics*, 50(5): 2002–2013.

Lin, X.; Baweja, H.; Kantor, G.; and Held, D. 2019. Adaptive auxiliary task weighting for reinforcement learning. *Advances in neural information processing systems*, 32.

Park, J.; Kim, T.; Seong, S.; and Koo, S. 2022. Control automation in the heat-up mode of a nuclear power plant using reinforcement learning. *Progress in Nuclear Energy*, 145: 104107.

Park, J. Y.; and Wong, L. 2022. Robust Imitation of a Few Demonstrations with a Backwards Model. *Advances in Neural Information Processing Systems*, 35: 19759–19772.

Radaideh, M. I.; Wolverton, I.; Joseph, J.; Tusar, J. J.; Otgonbaatar, U.; Roy, N.; Forget, B.; and Shirvan, K. 2021. Physics-informed reinforcement learning optimization of nuclear assembly design. *Nuclear Engineering and Design*, 372: 110966.

Rafiee, B.; Jin, J.; Luo, J.; and White, A. 2022. What makes useful auxiliary tasks in reinforcement learning: investigating the effect of the target policy. *arXiv preprint arXiv:2204.00565*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*.

Sutton, R. S. 1984. *Temporal Credit Assignment in Reinforcement Learning*. Ph.D. thesis.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*.

Tesauro, G. 1992. Practical Issues in Temporal Difference Learning. *Mach. Learn.*

van Hasselt, H.; Madjiheurem, S.; Hessel, M.; Silver, D.; Barreto, A.; and Borsa, D. 2020. Expected Eligibility Traces. *CoRR*.

Wang, J.; Li, W.; Jiang, H.; Zhu, G.; Li, S.; and Zhang, C. 2021. Offline reinforcement learning with reverse model-based imagination. *Advances in Neural Information Processing Systems*, 34: 29420–29432.

Zhang, S.; Veeriah, V.; and Whiteson, S. 2020. Learning Retrospective Knowledge with Reverse Reinforcement Learning. In *Advances in Neural Information Processing Systems*.