GRADIENT DESCENT TEMPORAL DIFFERENCE-DIFFERENCE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Off-policy learning algorithms, in which an agent updates the value function of the optimal policy while selecting actions using an independent exploration policy, provide an effective solution to the explore-exploit tradeoff and have proven to be of great practical value in reinforcement learning. However, even for simple convex problems such as linear value function approximation, these algorithms are not guaranteed to be stable. To address this, alternative algorithms that are provably convergent in such cases have been introduced, the most well known being gradient descent temporal difference (GTD) learning. This algorithm and others like it, however, tend to converge much more slowly than conventional temporal difference learning. In this paper we propose *gradient descent temporal difference-difference (Gradient-DD) learning* in order to accelerate GTD learning by introducing second-order differences in successive parameter updates. We investigate this algorithm in the framework of linear value function approximation, analytically showing its improvement over GTD learning. Studying the model empirically on the random walk and Boyan-chain prediction tasks, we find substantial improvement over GTD learning and, in several cases, better performance even than conventional TD learning.

1 INTRODUCTION

Off-policy methods offer a general and powerful framework for dealing with the exploration-exploitation tradeoff in reinforcement learning. In such methods, the agent makes use of an independent exploration policy to select actions while learning the value function for the optimal policy. However, because off-policy methods learn a value function for a target policy given data due to a different behavior policy, they often exhibit greater variance in parameter updates and. When applied to problems involving function approximation, off-policy methods are slower to converge than on-policy methods and may even diverge (Baird, 1995; Sutton & Barto, 2018).

Two general approaches have been investigated to address the challenge of developing stable and effective off-policy temporal-difference algorithms. One approach is to use importance sampling methods to warp the update distribution back to the on-policy distribution (Precup et al., 2000; Mahmood et al., 2014). This approach is useful for decreasing the variance of parameter updates, but it does not address stability issues. The second main approach to addressing the challenge of off-policy learning is to develop true gradient descent-based methods that are guaranteed to be stable regardless of the update distribution. Sutton et al. (2009a;b) proposed the first off-policy gradient-descent-based temporal difference (GTD and GTD2, respectively) algorithms. These algorithms are guaranteed to be stable, with computational complexity scaling linearly with the size of the function approximator. Empirically, however, their convergence is much slower than conventional temporal difference (TD) learning, limiting their practical utility. Building on this work, extensions to the GTD family of algorithms (see Ghiassian et al. (2018) for a review) have allowed for incorporating eligibility traces (Maei & Sutton, 2010; Geist & Scherrer, 2014), non-linear function approximation such as with a neural network (Maei, 2011), and reformulation of the optimization as a saddle point problem (Liu et al., 2015; Du et al., 2017). However, due to their slow convergence, none of these stable off-policy methods are commonly used in practice.

In this work, we introduce a new gradient descent algorithm for temporal difference learning with linear value function approximation. This algorithm, which we call *gradient descent temporal*

difference-difference (Gradient-DD) learning, is an acceleration technique that employs second-order differences in successive parameter updates. The basic idea of Gradient-DD is to modify the error objective function by additionally considering the prediction error obtained in last time step, then to derive a gradient-descent algorithm based on this modified objective function. In addition to exploiting the Bellman equation to get the solution, this modified error objective function avoids drastic changes in the value function estimate by encouraging local search around the current estimate. Algorithmically, the Gradient-DD approach only adds an additional term to the update rule of the GTD2 method, and the extra computational cost is negligible. We show mathematically that applying this method significantly improves the convergence rate relative to the GTD2 method for linear function approximation. This result is supported by numerical experiments, which also show that Gradient-DD obtains better convergence in many cases than conventional TD learning.

2 GRADIENT DESCENT METHOD FOR OFF-POLICY TEMPORAL DIFFERENCE LEARNING

2.1 PROBLEM DEFINITION AND BACKGROUND

In this section, we formalize the problem of learning the value function for a given policy under the Markov Decision Process (MDP) framework. In this framework, the agent interacts with the environment over a sequence of discrete time steps, $t = 1, 2, \ldots$. At each time step the agent observes a partial summary of the state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$. In response, the environment emits a reward $r_t \in \mathbb{R}$ and transitions the agent to its next state $s_{t+1} \in \mathcal{S}$. The state and action sets are finite. State transitions are stochastic and dependent on the immediately preceding state and action. Rewards are stochastic and dependent on the preceding state and action, as well as on the next state. The process generating the agent's actions is termed the behavior policy. In off-policy learning, this behavior policy is in general different from the target policy $\pi : \mathcal{S} \to \mathcal{A}$, which we assume to be deterministic. The objective is to learn an approximation to the state-value function under the target policy in a particular environment:

$$V(s) = \mathrm{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_1 = s \right], \tag{1}$$

where $\gamma \in [0, 1)$ is the discount rate.

In problems for which the state space is large, it is practical to approximate the value function. In this paper we consider linear function approximation, where states are mapped to feature vectors with fewer components than the number of states. Specifically, for each state $s \in \mathcal{S}$ there is a corresponding feature vector $\mathbf{x}(s) \in \mathbb{R}^p$, with $p \leq |\mathcal{S}|$, such that the approximate value function is given by

$$V_{\mathbf{w}}(s) := \mathbf{w}^\top \mathbf{x}(s). \tag{2}$$

The goal is then to learn the parameters $\mathbf{w}$ such that $V_{\mathbf{w}}(s) \approx V(s)$.

2.2 GRADIENT TEMPORAL DIFFERENCE LEARNING

A major breakthrough for the study of the convergence properties of MDP systems came with the introduction of the GTD and GTD2 learning algorithms (Sutton et al., 2009a;b). We begin by briefly recapitulating the GTD algorithms, which we will then extend in the following sections. To begin, we introduce the Bellman operator $\mathbf{B}$ such that the true value function $\mathbf{V} \in \mathbb{R}^{|\mathcal{S}|}$ satisfies the Bellman equation:

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P} \mathbf{V} =: \mathbf{B} \mathbf{V},$$

where $\mathbf{R}$ is the reward vector with components $\mathrm{E}(r_{n+1}|s_n = s)$, and $\mathbf{P}$ is a matrix of state transition probabilities. In temporal difference methods, an appropriate objective function should minimize the difference between the approximate value function and the solution to the Bellman equation.

Having defined the Bellman operator, we next introduce the projection operator $\mathbf{\Pi}$, which takes any value function $\mathbf{V}$ and projects it to the nearest value function within the space of approximate value functions of the form (2). Letting $\mathbf{X}$ be the matrix whose rows are $\mathbf{x}(s)$, the approximate value function can be expressed as $\mathbf{V}_{\mathbf{w}} = \mathbf{X}\mathbf{w}$. We will also assume that there exists a limiting probability

distribution such that $d_s = \lim_{n\to\infty} p(s_n = s)$ (or, in the episodic case, d_s is the proportion of time steps spent in state s). The projection operator is then given by

$$\mathbf{\Pi} = \mathbf{X}(\mathbf{X}^\top \mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{D},$$

where the matrix $\mathbf{D}$ is diagonal, with diagonal elements d_s.

The natural measure of how closely the approximation $\mathbf{V_w}$ satisfies the Bellman equation is the mean-squared Bellman error:

$$\text{MSBE}(\mathbf{w}) = \|\mathbf{V_w} - \mathbf{B}\mathbf{V_w}\|^2_{\mathbf{D}}, \tag{3}$$

where the norm is weighted by $\mathbf{D}$, such that $\|\mathbf{V}\|^2_{\mathbf{D}} = \mathbf{V}^\top\mathbf{D}\mathbf{V}$. However, because the Bellman operator follows the underlying state dynamics of the Markov chain, irrespective of the structure of the linear function approximator, $\mathbf{B}\mathbf{V_w}$ will typically not be representable as $\mathbf{V_w}$ for any $\mathbf{w}$. An alternative objective function, therefore, is the mean squared *projected* Bellman error (MSPBE), which we define as

$$J(\mathbf{w}) = \|\mathbf{V_w} - \mathbf{\Pi}\mathbf{B}\mathbf{V_w}\|^2_{\mathbf{D}}. \tag{4}$$

Following (Sutton et al., 2009b), our objective is to minimize this error measure. As usual in stochastic gradient descent, the weights at each time step are then updated by $\Delta\mathbf{w} = -\alpha\nabla_{\mathbf{w}}J(\mathbf{w})$, where $\alpha > 0$, and

$$\begin{aligned} -\frac{1}{2}\nabla_{\mathbf{w}}J(\mathbf{w}) =& -\mathrm{E}[(\gamma\mathbf{x}_{n+1} - \mathbf{x}_n)\mathbf{x}_n^\top][\mathrm{E}(\mathbf{x}_n\mathbf{x}_n^\top)]^{-1}\mathrm{E}(\delta_n\mathbf{x}_n) \\ \approx& -\mathrm{E}[(\gamma\mathbf{x}_{n+1} - \mathbf{x}_n)\mathbf{x}_n^\top]\boldsymbol{\eta}. \end{aligned} \tag{5}$$

For notational simplicity, we have denoted the feature vector associated with s_n as $\mathbf{x}_n = \mathbf{x}(s_n)$. We have also introduced the temporal difference error $\delta_n = r_n + (\gamma\mathbf{x}_{n+1} - \mathbf{x}_n)^\top\mathbf{w}_n$, as well as $\boldsymbol{\eta}$, a linear predictor to approximate $[\mathrm{E}(\mathbf{x}_n\mathbf{x}_n^\top)]^{-1}\mathrm{E}(\delta_n\mathbf{x}_n)$. Because the factors in Eqn. (5) can be directly sampled, the resulting updates in each step are

$$\begin{aligned} \delta_n =& r_n + (\gamma\mathbf{x}_{n+1} - \mathbf{x}_n)^\top\mathbf{w}_n \\ \boldsymbol{\eta}_{n+1} =& \boldsymbol{\eta}_n + \beta_n(\delta_n - \mathbf{x}_n^\top\boldsymbol{\eta}_n)\mathbf{x}_n \\ \mathbf{w}_{n+1} =& \mathbf{w}_n - \alpha_n(\gamma\mathbf{x}_{n+1} - \mathbf{x}_n)(\mathbf{x}_n^\top\boldsymbol{\eta}_n). \end{aligned} \tag{6}$$

These updates define the GTD2 learning algorithm, which we will build upon in the following section.

3 GRADIENT DESCENT TEMPORAL DIFFERENCE-DIFFERENCE LEARNING

In order to accelerate the GTD2 algorithm described above, in this section we modify the objective function via additionally considering the approximation error $\mathbf{V_w} - \mathbf{V}_{\mathbf{w}_{n-1}}$ given the previous time step $n-1$. Specifically, we modify Eqn. (4) as follows:

$$J_{\text{GDD}}(\mathbf{w}|\mathbf{w}_{n-1}) = J(\mathbf{w}) + \kappa\|\mathbf{V_w} - \mathbf{V}_{\mathbf{w}_{n-1}}\|^2_{\mathbf{D}}, \tag{7}$$

where $\kappa \geq 0$ is a parameter of the regularization.

Minimizing Eqn. (7) is equivalent to the following optimization

$$\arg\min_{\mathbf{w}} J(\mathbf{w}) \text{ s.t. } \|\mathbf{V_w} - \mathbf{V}_{\mathbf{w}_{n-1}}\|^2_{\mathbf{D}} \leq \mu \tag{8}$$

where $\mu > 0$ is a parameter which becomes large when κ is small, so that the MSPBE objective is recovered as $\mu \to \infty$, equivalent to $\kappa \to 0$ in Eqn. (7). We show in the Appendix that for any $\mu > 0$, there exist $\kappa \geq 0$ such that the solution of Eqn. (7) and that of Eqn. (8) are the same.

Eqns. (7) and (8) represent a tradeoff between minimizing the MSPBE error and preventing the estimated value function from changing too drastically. Rather than simply minimizing the optimal prediction from the projected Bellman equation, the agent makes use of the most recent update to look for the solution. Figure 1 gives a schematic view of the effect of the regularization. Rather than directly following the direction of the MSPBE gradient, the update chooses a $\mathbf{w}$ that minimizes the MSPBE while following the constraint that the estimated value function should not change too

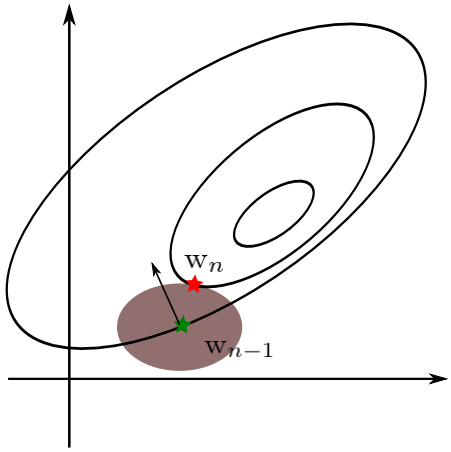


Figure 1: Schematic diagram of Gradient-DD learning with $\mathbf{w} \in \mathbb{R}^2$. Rather than updating $\mathbf{w}$ directly along the gradient of the MSPBE (arrow), the update rule selects $\mathbf{w}_n$ that minimizes the MSPBE while satisfying the constraint $\|\mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}_{n-1}}\|_{\mathbf{D}}^2 \leq \mu$ (shaded ellipse).

greatly. In effect, the regularization term encourages re-experience around the estimate at previous time step, especially when the state space is large. This is similar to the trust region policy optimization algorithm (Peters & Schaal, 2008; Schulman et al., 2015), in which the constrained optimization is used to update the policy by considering the constraint on the KL divergence (or some other measure) between the new policy and the old policy. Here, however, our aim is to look for the optimal value function $\mathbf{V}_{\mathbf{w}}$, and the regularization term uses the previous value function estimate to avoid large biases in the updating process.

With these considerations in mind, the negative gradient of $J_{\text{GDD}}(\mathbf{w}|\mathbf{w}_{n-1})$ is

$$\begin{aligned}
&-\frac{1}{2}\nabla_{\mathbf{w}} J_{\text{GDD}}(\mathbf{w}|\mathbf{w}_{n-1}) \\
=&-\mathrm{E}[(\gamma\mathbf{x}_{n+1}-\mathbf{x}_n)\mathbf{x}_n^\top][\mathrm{E}(\mathbf{x}_n\mathbf{x}_n^\top)]^{-1}\mathrm{E}(\delta_n\mathbf{x}_n) - \kappa\mathrm{E}[(\mathbf{x}_n^\top\mathbf{w}_n - \mathbf{x}_n^\top\mathbf{w}_{n-1})\mathbf{x}_n] \\
\approx&-\mathrm{E}[(\gamma\mathbf{x}_{n+1}-\mathbf{x}_n)\mathbf{x}_n^\top]\boldsymbol{\eta}_n - \kappa\mathrm{E}[(\mathbf{x}_n^\top\mathbf{w}_n - \mathbf{x}_n^\top\mathbf{w}_{n-1})\mathbf{x}_n]. \qquad (9)
\end{aligned}$$

Because the terms in Eqn. (9) can be directly sampled, the stochastic gradient descent updates are given by

$$\begin{aligned}
\delta_n =& r_n + (\gamma\mathbf{x}_{n+1}-\mathbf{x}_n)^\top\mathbf{w}_n \\
\boldsymbol{\eta}_{n+1} =& \boldsymbol{\eta}_n + \beta_n(\delta_n - \mathbf{x}_n^\top\boldsymbol{\eta}_n)\mathbf{x}_n \\
\mathbf{w}_{n+1} =& \mathbf{w}_n - \kappa_n(\mathbf{x}_n^\top\mathbf{w}_n - \mathbf{x}_n^\top\mathbf{w}_{n-1})\mathbf{x}_n - \alpha_n(\gamma\mathbf{x}_{n+1}-\mathbf{x}_n)(\mathbf{x}_n^\top\boldsymbol{\eta}_n). \qquad (10)
\end{aligned}$$

These update equations define the Gradient-DD method, in which the GTD2 update equations (6) are generalized by including a second-order update term in the third update equation, where this term originates from the squared bias term in the objective (6). In the following sections, we shall analytically and numerically investigate the convergence and performance of Gradient-DD learning.

4 Improved convergence rate

In this section we analyze the convergence rate of Gradient-DD learning. Note that the second-order update in the last line in Eqn. (10) can be rewritten as a system of first-order difference equations:

$$\begin{aligned}
(\mathbf{I} + \kappa_n\mathbf{x}_n\mathbf{x}_n^\top)(\mathbf{w}_{n+1} - \mathbf{w}_n) =& \kappa_n\mathbf{x}_n\mathbf{x}_n^\top(\mathbf{u}_{n+1} - \mathbf{u}_n) - \alpha_n(\gamma\mathbf{x}_{n+1} - \mathbf{x}_n)(\mathbf{x}_n^\top\boldsymbol{\eta}_n); \\
\mathbf{u}_{n+1} =& \mathbf{w}_{n+1} - \mathbf{w}_n. \qquad (11)
\end{aligned}$$

Let $\beta_n = \zeta\alpha_n$, $\zeta > 0$. We consider the constant step sizes in the updates, i.e., $\kappa_n = \kappa$ and $\alpha_n = \alpha$. Denote $\mathbf{H}_n = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{x}_n\mathbf{x}_n^\top \end{bmatrix}$ and $\mathbf{G}_n = \begin{bmatrix} \sqrt{\zeta}\mathbf{x}_n\mathbf{x}_n^\top & \mathbf{x}_n(\mathbf{x}_n - \gamma\mathbf{x}_{n+1})^\top \\ -(\mathbf{x}_n - \gamma\mathbf{x}_{n+1})\mathbf{x}_n^\top & \mathbf{0} \end{bmatrix}$. We rewrite the update rules of two iterations in Eqn. (11) as a single iteration in a combined parameter vector with $2n$ components, $\boldsymbol{\rho}_n = (\boldsymbol{\eta}_n^\top/\sqrt{\zeta}, \mathbf{w}_n^\top)^\top$, and a new reward-related vector with $2n$ components, $\mathbf{g}_{n+1} = (r_n\mathbf{x}_n^\top, \mathbf{0}^\top)^\top$, as follows:

$$\boldsymbol{\rho}_{n+1} = \boldsymbol{\rho}_n - \kappa\mathbf{H}_n(\boldsymbol{\rho}_n - \boldsymbol{\rho}_{n-1}) - \sqrt{\zeta}\alpha(\mathbf{G}_n\boldsymbol{\rho}_n - \mathbf{g}_{n+1}), \qquad (12)$$

Denoting $\boldsymbol{\psi}_{n+1} = \alpha^{-1}(\boldsymbol{\rho}_{n+1} - \boldsymbol{\rho}_n)$, Eqn. (12) is rewritten as

$$\begin{aligned}\begin{bmatrix} \boldsymbol{\rho}_{n+1} - \boldsymbol{\rho}_n \\ \boldsymbol{\psi}_{n+1} - \boldsymbol{\psi}_n \end{bmatrix} =& \alpha \begin{bmatrix} \mathbf{I} + \kappa \mathbf{H}_n & -\kappa\alpha \mathbf{H}_n \\ \mathbf{I} & -\alpha \mathbf{I} \end{bmatrix}^{-1} \begin{bmatrix} -\sqrt{\zeta}(\mathbf{G}_n \boldsymbol{\rho}_n - \mathbf{g}_{n+1}) \\ \boldsymbol{\psi}_n \end{bmatrix} \\ =& \alpha \begin{bmatrix} -\sqrt{\zeta}\mathbf{G}_n & -\kappa \mathbf{H}_n \\ -\sqrt{\zeta}\alpha^{-1}\mathbf{G}_n & -\alpha^{-1}(\mathbf{I} + \kappa \mathbf{H}_n) \end{bmatrix} \begin{bmatrix} \boldsymbol{\rho}_n \\ \boldsymbol{\psi}_n \end{bmatrix} + \alpha \begin{bmatrix} \sqrt{\zeta}\mathbf{g}_{n+1} \\ \sqrt{\zeta}\alpha^{-1}\mathbf{g}_{n+1} \end{bmatrix},\end{aligned} \tag{13}$$

where the second step is from $\begin{bmatrix} \mathbf{I} + \kappa \mathbf{H}_n & -\kappa\alpha \mathbf{H}_n \\ \mathbf{I} & -\alpha \mathbf{I} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{I} & -\kappa \mathbf{H}_n \\ \alpha^{-1}\mathbf{I} & -\alpha^{-1}(\mathbf{I} + \kappa \mathbf{H}_n) \end{bmatrix}$. Denote $\mathbf{J}_n = \begin{bmatrix} -\sqrt{\zeta}\mathbf{G}_n & -\kappa \mathbf{H}_n \\ -\sqrt{\zeta}\alpha^{-1}\mathbf{G}_n & -\alpha^{-1}(\mathbf{I} + \kappa \mathbf{H}_n) \end{bmatrix}$. Eqn. (13) tells us that $\mathbf{J}_n$ is the update matrix of the Gradient-DD algorithm. Note: $\mathbf{G}_n$ is the update matrix of the GTD2 algorithm. Therefore, the improved convergence rate of Gradient-DD learning can be understood by considering the real eigenvalues of the matrices $\mathrm{E}(\mathbf{G}_n)$ denoted by $\mathbf{G}$, and $\mathrm{E}(\mathbf{J}_n)$ denoted by $\mathbf{J}$.

Obviously, $\mathbf{J} = \begin{bmatrix} -\sqrt{\zeta}\mathbf{G} & -\kappa \mathbf{H} \\ -\sqrt{\zeta}\alpha^{-1}\mathbf{G} & -\alpha^{-1}(\mathbf{I} + \kappa \mathbf{H}) \end{bmatrix}$, where $\mathbf{H} = \mathrm{E}(\mathbf{H}_n)$. Let λ_G be a real eigenvalue of the matrix $\sqrt{\zeta}\mathbf{G}$, and let λ be an eigenvalue of the matrix $\mathbf{J}$. The latter must be a solution to

$$|\lambda \mathbf{I} - \mathbf{J}| = (\lambda + \alpha^{-1})(\lambda + \lambda_G) + \kappa\alpha^{-1} = \lambda^2 + (\alpha^{-1} + \lambda_G)\lambda + \alpha^{-1}(\lambda_G + \kappa) = 0. \tag{14}$$

In the case where $0 < \kappa < \alpha(\lambda_G - \alpha^{-1})^2/4$, the solutions to Eqn. (14) are

$$\lambda = -\frac{1}{2}(\alpha^{-1} + \lambda_G) \pm \frac{1}{2}\sqrt{(\alpha^{-1} + \lambda_G)^2 - 4\alpha^{-1}(\lambda_G + \kappa)} < -\lambda_G.$$

It follows that the effect of the second-order difference term is to accelerate the effect of the update matrix. This explains the enhanced speed of convergence in Gradient-DD learning. We shall illustrate this enhanced speed of convergence in numerical experiments in Section 5.

5 Empirical Study

In this section, we assess the practical utility of the Gradient-DD method in numerical experiments. To validate performance of Gradient-DD learning, we compare Gradient-DD learning with GTD2 learning, TDC learning (TD with gradient correction (Sutton et al., 2009b)), TD learning, and Emphatic TD learning (Sutton & Mahmood, 2016) in tabular representation using a random-walk task and in linear representation using the Boyan-chain task. For each method and each task, we performed a scan over the step sizes α_n and κ_n so that the comprehensive performance of the different algorithms can be compared. We considered three choices of step size sequence $\{\alpha_n, \kappa_n\}$:

- (Case 1) Both α_n and κ_n are constant, i.e., $\alpha_n = \alpha_0$, and $\kappa_n = c\alpha_0$ where $c = 1, 2, 4$.
- (Case 2) The learning rate α_n is tapered according to the schedule $\alpha_n = \alpha_0(10^3 + 1)/(10^3 + n)$. Again we set $\kappa_n = c\alpha_n$.

Additionally, we also consider Case 3: α_n is tapered as in Case 2, but κ_n is constant, with $\kappa_n = c\alpha_0$. In order to simplify presentation, the results of Case 3 are reported in the Appendix.

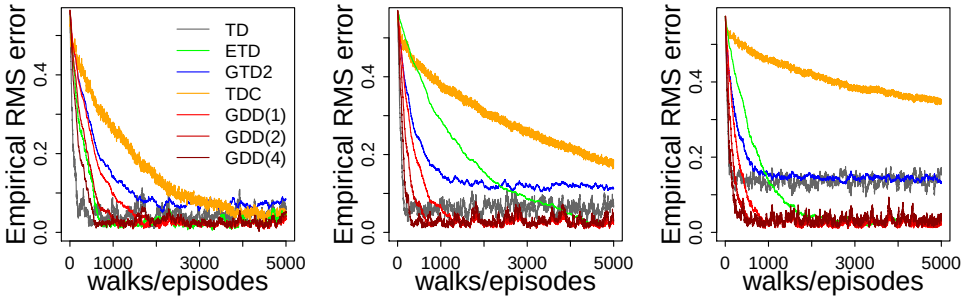
Because Emphatic TD, unlike Gradient-based TD, requires the step size α to be especially small, the plotted values of α_0 for this method are tuned relative to the values used in the algorithm defined in (Sutton & Mahmood, 2016), where α_0 is chosen from $\{0.5\%, 0.1\%, 0.05\%, 0.01\%\}$ for the random-walk task or from $\{0.5\%, 0.2\%, 0.1\%, 0.05\%\}$ for the Boyan-chain task by the best performance at the final episode. In all cases, we set $\gamma = 1$.

5.1 Random walk task

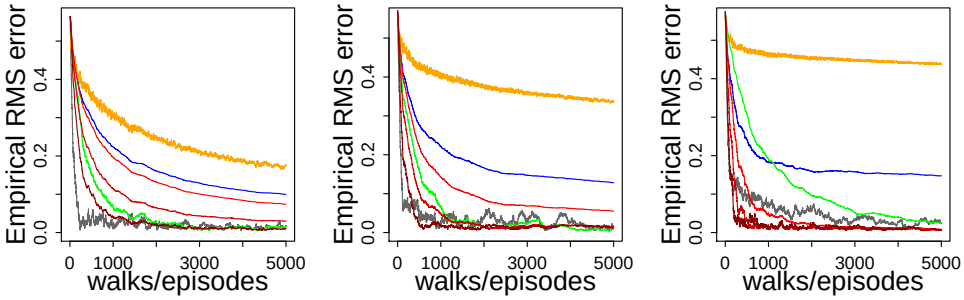
As a first test of Gradient-DD learning, we conducted a simple random walk task (Sutton & Barto, 2018) with tabular representation of the value function. The random walk task has a linear arrangement of m states plus an absorbing terminal state at each end. Thus there are $m+2$ sequential states,

$S_0, S_1, \cdots, S_m, S_{m+1}$, where $m = 20$, 50, or 100. Every walk begins in the center state. At each step, the walk moves to a neighboring state, either to the right or to the left with equal probability. If either edge state (S_0 or S_{m+1}) is entered, the walk terminates. A walk's outcome is defined to be $r = 0$ at S_0 and $r = 1$ at S_{m+1}. Our aim is to learn the value of each state $V(s)$, where the true values are $(1, \cdots, m)/(m+1)$. In all cases the approximate value function is initialized to the intermediate value $V_0(s) = 0.5$.

We first look closely at the performance during training, which we show in Figure 2, where each method and parameter setting was run for 5000 episodes. From the figure we draw several observations. (1) For all conditions tested, Gradient-DD converges much more rapidly than GTD2 and TDC. The results indicate that Gradient-DD even converges faster than TD learning in some cases, though it is not as fast in the beginning episodes. (2) The advantage of Gradient-DD learning over other methods grows as the state space increases in size. (3) Gradient-DD learning is robust to the choice of c, which controls the size κ of the second-order update, as long as c is not too large. (Empirically $c = 2$ is a good choice.) (4) Gradient-DD has consistent and good performance under both the constant step size setting and under the tapered step size setting. In summary, compared with the GTD2 learning, Gradient-DD learning in this task leads to accelerated learning with good convergence.



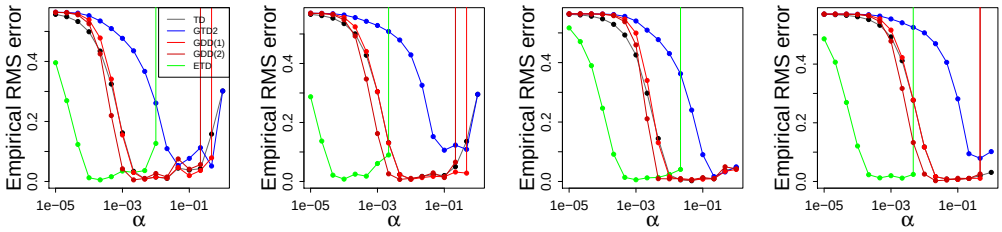
(a) Constant step size $\alpha_n = \alpha_0$ and $\kappa_n = c\alpha_0$.



(b) Tapering step size $\alpha_n = \alpha_0(10^3 + 1)/(10^3 + n)$ and $\kappa_n = c\alpha_n$.

Figure 2: Performance of Gradient-DD in the random walk task. From left to right in each subfigure: the size of state space is 10 ($\alpha_0 = 0.1$), 20 ($\alpha_0 = 0.2$), 50 ($\alpha_0 = 0.5$).

Next, we compare the methods by plotting the empirical RMS error from the final episode during training as a function of step size α in Figure 3, where 10000 episodes are used. Note: TDC is not reported here due to the performance shown in Figure 2. From the figure, we can make several observations. (1) Emphatic TD works well but is sensitive to α. It prefers very small α even in the tapering case, and this preference becomes strong as the state space becomes large in size. (2) Gradient-DD works well and is robust to α as the conventional TD learning does, while the GTD2 method works worse than conventional TD learning. This advantage is consistent in different settings.



(a) Constant step size. (b) Tapering step size.

Figure 3: Performance in the random walk task depends on step size. (a), Constant step size $\alpha_n = \alpha_0$ and $\kappa_n = c\alpha_0$. (b), Tapering step size $\alpha_n = \alpha_0(10^3 + 1)/(10^3 + n)$ and $\kappa_n = c\alpha_n$. In (a) and (b), state space size 10 (left) or 20 (right). GDD(c) denotes the Gradient-DD with c.

5.2 Boyan-chain task

We next investigate Gradient-DD learning on the Boyan-chain problem, which is a standard task for testing linear value-function approximation (Boyan, 2002). In this task we allow for $4p - 3$ states, with $p = 20$, each of which is represented by a p-dimensional feature vector. The p-dimensional representation for every fourth state from the start is $[1, 0, \cdots, 0]$ for state s_1, $[0, 1, 0, \cdots, 0]$ for s_5, $\cdots$, and $[0, 0, \cdots, 0, 1]$ for the terminal state s_{4p-3}. The representations for the remaining states are obtained by linearly interpolating between these. The optimal coefficients of the feature vector are $(-4(p-1), -4(p-2), \cdots, 0)/5$. Simulations with $p = 50$ and 100 give similar results to those from the random walk task, and hence are not shown here. In each state, except for the last one before the end, there are two possible actions: move forward one step or move forward two steps with equal probability 0.5. Both actions lead to reward -0.3. The last state before the end just has one action of moving forward to the terminal with reward -0.2.

We report the results in Figure 4, which leads to conclusions similar to those already drawn from Figure 2. (1) Gradient-DD has much faster convergence than GTD2 and TDC, and generally converges to better values despite being somewhat slower than TD learning at the beginning episodes. (2) Gradient-DD is competitive with Emphatic TD. The improvement over other methods grows as the state space becomes larger. (3) As κ increases, the performance of Gradient-DD improves. Additionally, the performance of Gradient-DD is robust to changes in κ as long as κ is not very large. Empirically a good choice is to set $\kappa = \alpha$ or 2α. (4) Comparing the performance with constant step size versus that with tapered step size, the Gradient-DD method performs better with tapered step size than it does with constant step size.

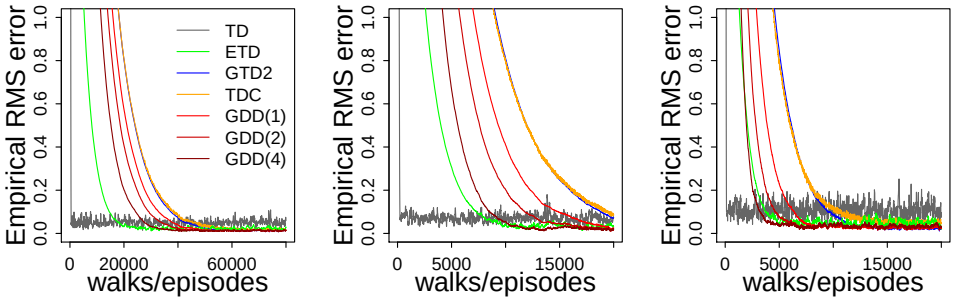
5.3 Baird's counterexample

We also verify the performance of Gradient-DD on Baird's off-policy counterexample (Baird, 1995), for which TD learning famously diverges and TDC is not reported here due to its similarity to GTD2. We consider two cases: one is 7-state, and the other is 100-state. We set $\alpha = 0.02$ (but $\alpha = 10^{-5}$ for ETD), $\kappa = 10\alpha$, $\eta = 1$ and $\gamma = 0.99$. For the initial parameter values $(1, \cdots, 1, 10, 1)^\top$. We measure the performance by the empirical RMS errors as function of sweep, and report the results in Figure 5. The figure demonstrates that Gradient-DD works as well on this well-known counterexample as GTD2 does, and even works better than GTD2 for the 100-state case.

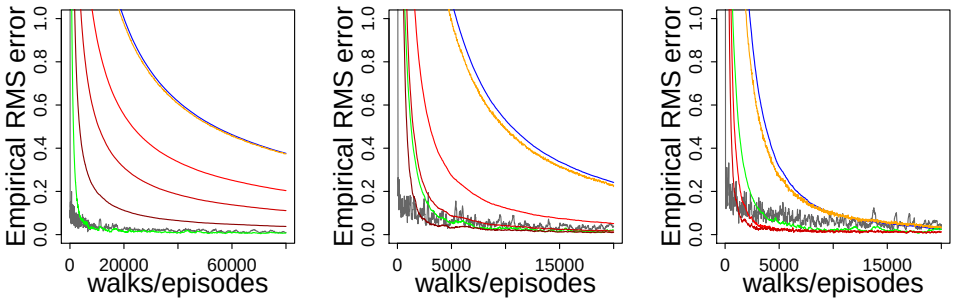
6 Conclusion and discussion

In this work, we have proposed Gradient-DD learning, a new gradient descent-based TD learning algorithm. The algorithm is based on a modification of the projected Bellman error objective function for value function approximation by introducing a second-order difference term. The algorithm significantly improves upon existing methods for gradient-based TD learning, obtaining better convergence performance than conventional linear TD learning.

Since GTD learning was originally proposed, the Gradient-TD family of algorithms has been extended for incorporating eligibility traces and learning optimal policies (Maei & Sutton, 2010; Geist



(a) Constant step size $\alpha_n = \alpha_0$, where $\alpha_0 = 0.05, 0.1, 0.2$ from left to right, and $\kappa_n = c\alpha_0$.



(b) Tapering step size $\alpha_n = \alpha_0(10^3 + 1)/(10^3 + n)$, where $\alpha_0 = 0.3, 0.5, 0.8$ from left to right, and $\kappa_n = c\alpha_n$.

Figure 4: Performance of Gradient-DD in the Boyan Chain task with 20-features. Note that the case Gradient-DD(4), i.e. $c = 4$, is not shown when it does not converge.

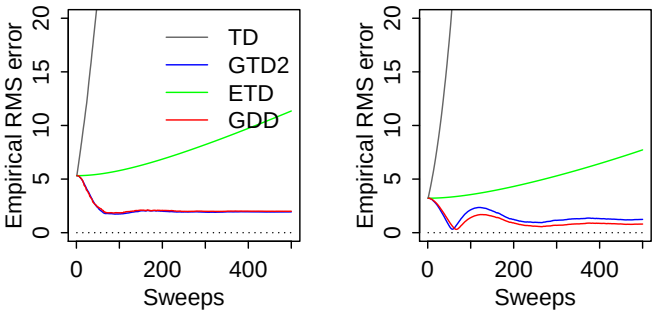


Figure 5: Bairds off-policy counterexample with two versions: 7-state (left) and 100-state (right).

& Scherrer, 2014), as well as for application to neural networks (Maei, 2011). Additionally, many variants of the vanilla Gradient-TD methods have been proposed, including HTD (Hackman, 2012) and Proximal Gradient-TD (Liu et al., 2016). Because Gradient-DD just modifies the objective error of GTD2 by considering an additional squared-bias term, it may be extended and combined with these other methods, potentially broadening its utility for more complicated tasks.

In this work we have focused on value function prediction in the two simple cases of tabular representations and linear approximation. An especially interesting direction for future study will be the application of Gradient-DD learning to tasks requiring more complex representations, including neural network implementations. Such approaches are especially useful in cases where state spaces are large, and indeed we have found in our results that Gradient-DD seems to confer the greatest advantage over other methods in such cases. Intuitively, we expect that this is because the difference between the optimal update direction and that chosen by gradient descent becomes greater in higher-dimensional spaces (cf. Fig. 1). This performance benefit in large state spaces suggests that Gradient-DD may be of practical use for these more challenging cases.

REFERENCES

L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12 th International Conference on Machine Learning*, pp. 30–37, 1995.

Justin A. Boyan. Technical update: least-squares temporal difference learning. *Machine Learning*, 49:233–246, 2002.

S. S. Du, J. Chen, L. Li, L. Xiao, and D. Zhou. Stochastic variance reduction methods for policy evaluation. In *Proceedings of the 34 th International Conference on Machine Learning*, 2017.

M. Geist and B. Scherrer. Off-policy learning with eligibility traces: A survey. *Journal of Machine Learning Research*, 15:289–333, 2014.

S. Ghiassian, A. Patterson, M. White, R.S. Sutton, and A. White. Online off-policy prediction. *CoRR*, 2018.

L. Hackman. Faster gradient-td algorithms. Master's thesis, University of Alberta, Edmonton, 2012.

B. Liu, J. Liu, M. Ghavamzadeh, S. Mahadevan, and M. Petrik. Finite-sample analysis of proximal gradient TD algorithms. In *Proceedings of the 31st International Conference on Uncertainty in Artificial Intelligence*, pp. 504–513, 2015.

B. Liu, J. Liu, M. Ghavamzadeh, S. Mahadevan, and M. Petrik. Proximal gradient temporal difference learning algorithms. In *The 25th International Conference on Arti cial Intelligence (IJCAI-16),*, 2016.

H.R. Maei. *Gradient temporal-difference learning algorithms*. PhD thesis, University of Alberta, Edmonton, 2011.

H.R. Maei and R.S. Sutton. GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the 3rd Conference on Artificial General Intelligence*, pp. 91–96, 2010.

A. R. Mahmood, H. van Hasselt, and R. S. Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems 27*, 2014.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

D. Precup, R. S. Sutton, and S. Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17 th International Conference on Machine Learning*, 2000.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

R. S. Sutton, Cs. Szepesvári, and H. R. Maei. A convergent O(n) algorithm for off-policy temporal difference learning with linear function approximation. In *Advances in Neural Information Processing Systems 21*, 2009a.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition edition, 2018.

R.S. Sutton and M. Mahmood, A.R. amd White. An emphatic approach to the problem of off-policy temporal difference learning. *Journal of Machine Learning Research*, 17(73):1–20, 2016.

R.S. Sutton, H.R. Maei, D. Precup, S. Bhatnagar, D. Silver, Cs. Szepesvári, and E. Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning*, 2009b.

APPENDIX

ON THE EQUIVALENCE OF EQNS. (7) & (8)

The Karush-Kuhn-Tucker conditions of Eqn. (8) are the following system of equations

$$\begin{cases} \frac{d}{d\mathbf{w}} J(\mathbf{w}) + \kappa \frac{d}{d\mathbf{w}} (\|\mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}_{n-1}}\|_{\mathbf{D}}^2 - \mu) = 0; \\ \kappa(\|\mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}_{n-1}}\|_{\mathbf{D}}^2 - \mu) = 0; \\ \|\mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}_{n-1}}\|_{\mathbf{D}}^2 \leq \mu; \\ \kappa \geq 0. \end{cases}$$

These equations are equivalent to

$$\begin{cases} \frac{d}{d\mathbf{w}} J(\mathbf{w}) + \kappa \frac{d}{d\mathbf{w}} \|\mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}_{n-1}}\|_{\mathbf{D}}^2 = 0 \text{ and } \kappa > 0, \\ \qquad\qquad\qquad\qquad \text{if } \|\mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}_{n-1}}\|_{\mathbf{D}}^2 = \mu; \\ \frac{d}{d\mathbf{w}} J(\mathbf{w}) = 0 \text{ and } \kappa = 0, \text{ if } \|\mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}_{n-1}}\|_{\mathbf{D}}^2 < \mu. \end{cases}$$

Thus, for any $\mu > 0$, there exists a $\kappa \geq 0$ such that $\frac{d}{d\mathbf{w}} J(\mathbf{w}) + \mu \frac{d}{d\mathbf{w}} \|\mathbf{V}_{\mathbf{w}} - \mathbf{V}_{\mathbf{w}_{n-1}}\|_{\mathbf{D}}^2 = 0$.

6.1 ADDITIONAL EMPIRICAL RESULTS

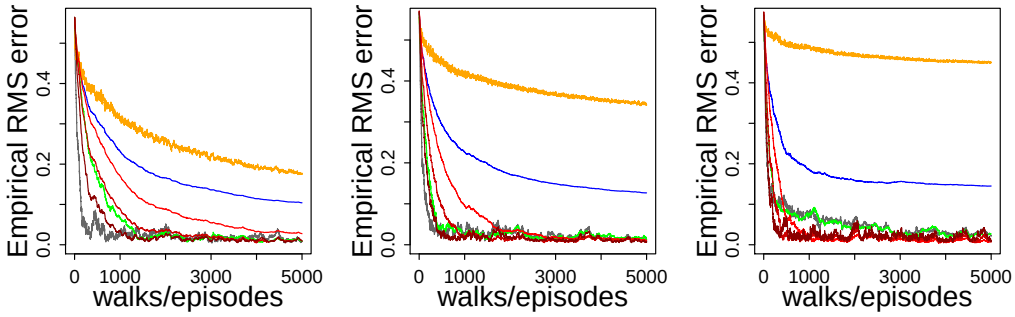


Figure 6: Performance of Gradient-DD in the random walk task under Case 3. Tapering step size $\alpha_n = \alpha_0(10^3 + 1)/(10^3 + n)$ and $\kappa_n = c\alpha_0$. From left to right in each subfigure: the size of state space is 10 ($\alpha_0 = 0.1$), 20 ($\alpha_0 = 0.2$), 50 ($\alpha_0 = 0.5$).

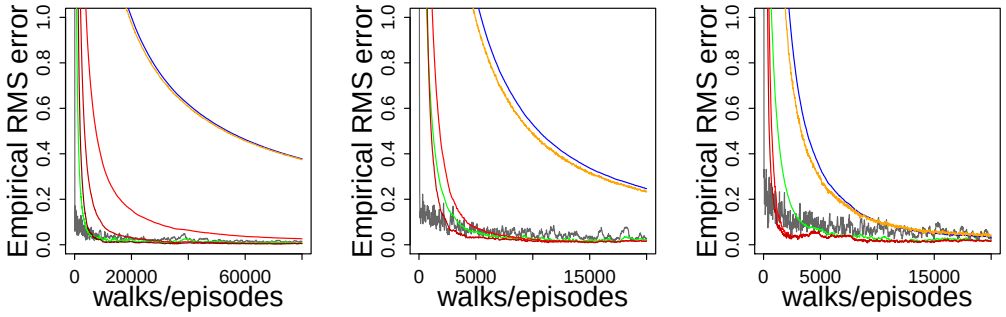


Figure 7: Performance of Gradient-DD in the Boyan Chain task with 20-features under Case 3. Tapering step size $\alpha_n = \alpha_0(10^3+1)/(10^3+n)$, where $\alpha_0 = 0.3, 0.5, 0.8,$, where $\alpha_0 = 0.3, 0.5, 0.8$ from left to right, and $\kappa_n = c\alpha_0$ Note that the case GDD(4), i.e. $c = 4$, is not shown when it does not converge.