Accelerated Gradient Temporal Difference Learning Algorithms

Dominik Meyer, Rémy Degenne[1], Ahmed Omrane[1] and Hao Shen
{dominik.meyer,remy.degenne,ahmed.omrane,hao.shen}@tum.de
Institute for Data Processing,
Technische Universität München, Germany

***Abstract*—In this paper we study Temporal Difference (TD) Learning with linear value function approximation. The classic TD algorithm is known to be unstable with linear function approximation and off-policy learning. Recently developed Gradient TD (GTD) algorithms have addressed this problem successfully. Despite their prominent properties of good scalability and convergence to correct solutions, they inherit the potential weakness of slow convergence as they are a stochastic gradient descent algorithm. Accelerated stochastic gradient descent algorithms have been developed to speed up convergence, while still keeping computational complexity low. In this work, we develop an accelerated stochastic gradient descent method for minimizing the Mean Squared Projected Bellman Error (MSPBE), and derive a bound for the Lipschitz constant of the gradient of the MSPBE, which plays a critical role in our proposed accelerated GTD algorithms. Our comprehensive numerical experiments demonstrate promising performance in solving the policy evaluation problem, in comparison to the GTD algorithm family. In particular, accelerated TDC surpasses state-of-the-art algorithms.**

I. Introduction

In Reinforcement Learning (RL), an agent interacting with its environment has the ultimate objective to optimize its policy based on the reward it gets from the taken actions. Policy iteration allows solving such a problem by iteratively evaluating the current policy and then improving it. Many approaches tackling the policy evaluation task have been developed. The Temporal Difference (TD) learning [9] approach is particularly known for being well appropriate for RL problems due to its temporal aspect.

Although the first TD algorithm, also known as TD(0), can achieve accurate predictions, it suffers from some major problems, since it is not guaranteed to converge under off-policy learning and non-linear function approximation. An alternative to TD(0) is the residual gradient algorithm introduced in [1]. This algorithm is considered to be the first gradient-based temporal difference algorithm. It overcomes the previous restrictions of TD(0), achieves smaller temporal differences but fails to attain the same TD(0) solution [6]. Recently, a new family of gradient TD algorithms was introduced in [10] and [11]. The three algorithms of this family (GTD, GTD2 and TDC) dealt with the deficiencies of TD(0) and guaranteed theoretical convergence to the same solution.

The gradient based algorithms, even though they have the virtue of being applicable to general settings with proved convergence properties, inevitably inherited the weaknesses of the stochastic gradient descent (SGD) that can harm the convergence speed of the algorithm. Historically, one of the approaches that allowed to remedy this issue is the accelerated gradient descent introduced in [7]. The importance of this approach comes from its computational simplicity since it only relies on first order information like the gradient descent method and does not increase the asymptotic computational cost. Consequently, this family of algorithms appealed to researchers from the machine learning community who were able to successfully implement it in several contexts, like the use of momentum methods in deep learning [8] or the extension of SGD to an accelerated version applicable to online learning [5].

The contribution of this paper is the extension of the gradient algorithms used in RL to new accelerated versions. In our work, we restricted ourselves to policy evaluation with linear function approximation for on-policy learning. We describe accelerated versions of stochastic gradient based algorithms and derive a bound for the important Lipschitz constant of the gradient which is needed as a parameter to the accelerated algorithms. To investigate the performance of the new algorithms compared to their original counterparts, we employ seven numerical experiments. Three versions of a random walk chain, the Boyan chain, two random MDPs and a continuous cart-pole balancing task.

II. Notations and Preliminaries

In this work, we consider a RL process as a Markov Decision Process (MDP), defined as a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is a set of possible states of the environment, $\mathcal{A}$ is a set of actions of the agent, $P\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ the conditional transition probabilities $P(s,a,s')$ over state transitions from state s to state s' given an action a, $r\colon \mathcal{S} \to \mathbb{R}$ is a reward function assigning immediate reward r to a state s, and $\gamma \in [0,1]$ is a discount factor.

A. GTD Learning with Linear Function Approximation

The goal of a RL agent is to learn a mapping from states to actions, i.e. a *policy* $\pi\colon \mathcal{S} \to \mathcal{A}$, which maximizes the value function $V^\pi\colon \mathcal{S} \to \mathbb{R}$ of a state s taking a policy π, defined as

$$V^\pi(s) := \mathbb{E}\left[\textstyle\sum_{t=0}^{\infty} \gamma^t r(s_t) | s_0 = s, \pi\right]. \tag{1}$$

[1]This paper is based on results of an interdisciplinary student project of Rémy Degenne and Ahmed Omrane.

It is well known that, for a given policy π, the value function V^{π} fulfills the *Bellman equation*, i.e.

$$V^{\pi}(s) = r(s) + \gamma \sum_{s'} P(s, \pi(s), s') V^{\pi}(s'). \tag{2}$$

The right hand side of Eq. (2) is often referred to as the *Bellman operator* for policy π, denoted by $\mathcal{T}V^{\pi}(s)$. In other words, the value function $V^{\pi}(s)$ is the fixed point of the Bellman operator $\mathcal{T}V^{\pi}(s)$, i.e. $V^{\pi}(s) = \mathcal{T}V^{\pi}(s)$.

When the state space is too large or infinite, exact representation of the value function is often practically unfeasible. Function approximation is thus of great demand for estimating the actual value function. A popular approach is to construct a set of features by the map $\phi\colon \mathcal{S} \to \mathbb{R}^k$, which are called the *features* or *basis functions*, and then to approximate the value function by a linear function. Specifically, for a given state s, the value function is approximated by

$$V(s) \approx (\phi(s))^{\top}\theta =: V_{\theta}, \tag{3}$$

where $\theta \in \mathbb{R}^k$ is a parameter vector. In the setting of TD learning, the parameter θ is updated at each time step t, i.e. for each state transition and the associated reward (s_t, r_t, s'_t). Here, we consider the simple one-step TD learning with linear function approximation, i.e. $\lambda = 0$ in the framework of TD(λ) learning. The parameter θ is updated as $\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi_t$, where $\alpha_t > 0$ is a sequence of step-size parameters, and δ_t is the simple TD error

$$\delta_t = r_t + \theta_t^{\top}\left(\gamma\phi'_t - \phi_t\right). \tag{4}$$

Note, that the TD error δ_t can be considered as a function of the parameter θ_t. By abuse of notation, in the rest of the paper, we denote

$$\delta_{\theta} = \delta(\theta) := r + \theta^{\top}\left(\gamma\phi' - \phi\right). \tag{5}$$

In order to find an optimal parameter θ^* via an optimization process, one has to define an appropriate objective function, which accurately measures the correctness of the current value function approximation, i.e. how far the current approximation is away from the actual TD solution. Motivated by the fact that the value function is the fixed point of the Bellman operator for a given policy, correctness of an approximation V_{θ} can be simply measured by the TD error itself, i.e. the *Mean Squared Bellman Error* (MSBE), which is defined as

$$MSBE(\theta) := \tfrac{1}{2}\left\|V_{\theta} - \mathcal{T}V_{\theta}\right\|_D^2, \tag{6}$$

where $D \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is an invertible diagonal matrix, whose components are the steady state distribution under the current policy.

Ideally, the minimum of the MSBE function admits a good value function approximation. Unfortunately, it is well known that, in practice, the performance of an approximation V_{θ} depends on the pre-selected feature space $\mathcal{F} := \left\{\Phi^{\top}\theta | \theta \in \mathbb{R}^k\right\}$, i.e. the span of the rows of $\Phi := \phi(\mathcal{S})$. By introducing the projector as

$$\Pi = \Phi^{\top}\left(\Phi D \Phi^{\top}\right)^{-1}\Phi D, \tag{7}$$

the so-called *Mean Squared Projected Bellman Error* (MSPBE) is often preferred

$$J\colon \mathbb{R}^k \to \mathbb{R}, \qquad \begin{aligned} J(\theta) &:= \tfrac{1}{2}\left\|V_{\theta} - \Pi\mathcal{T}V_{\theta}\right\|_D^2 \\ &= \tfrac{1}{2}\mathbb{E}[\delta_{\theta}\phi]^{\top}\mathbb{E}[\phi\phi^{\top}]^{-1}\mathbb{E}[\delta_{\theta}\phi]. \end{aligned} \tag{8}$$

Minimizing the MSPBE function finds a fixed point of the projected Bellman operator in the feature space $\mathcal{F}$, i.e. $V_{\theta} = \Pi\mathcal{T}V_{\theta}$. By computing the gradient of $J(\theta)$ as

$$\nabla_J(\theta) = \mathbb{E}\big[(\gamma\phi'_t - \phi_t)\phi_t^{\top}\big]\left(\mathbb{E}\big[\phi_t\phi_t^{\top}\big]\right)^{-1}\mathbb{E}\big[\delta_t\phi_t\big], \tag{9}$$

a gradient descent algorithm can be formulated straightaway. Unfortunately, the evaluation of the gradient $\nabla_J(\theta)$ requires a triple independent sampling. In order to avoid this difficulty, a quasi-stationary estimate of $w \approx \left(\mathbb{E}\big[\phi_t\phi_t^{\top}\big]\right)^{-1}\mathbb{E}\big[\delta_t\phi_t\big]$ is proposed, i.e.

$$w_{t+1} = w_t + \beta_t(\delta_t - \phi_t^{\top}w_t)\phi_t, \tag{10}$$

which leads to two stochastic gradient descent TD algorithms, cf. [11].

The update rules of the resulted GTD2 and TDC algorithms are specified by

$$\theta_{t+1} = \theta_t + \alpha_t(\phi_t - \gamma\phi'_t)(\phi_t^{\top}w_t) \tag{11}$$

and

$$\theta_{t+1} = \theta_t + \alpha_t(\delta_t\phi_t - \gamma(\phi_t^{\top}w_t)\phi'_t), \tag{12}$$

respectively. Here α_t and β_t are appropriate step sizes. It has been proven that both algorithms are asymptotically convergent to the correct TD solution. However, as typical stochastic gradient descent algorithms, the convergence rate of both is still in $O(\frac{1}{t})$. In order to achieve higher order of convergence, we propose to apply the Nesterov's acceleration strategy to minimize the MSPBE.

B. Nesterov's Accelerated Gradient Descent Algorithm

Let us consider a problem of minimizing a smooth convex cost function

$$f\colon \mathbb{R}^m \to \mathbb{R}. \tag{13}$$

A standard gradient descent algorithm, defined as the following iteration

$$x_{t+1} = x_t - \gamma\nabla_f(x_t), \tag{14}$$

where $\nabla_f(x_t)$ is the gradient of $f(x)$ at x_t and $\gamma > 0$ is an appropriate step size, converges to a minimum of $f(x)$. It is well known that such a gradient decent algorithm has only a convergence rate of $O(\frac{1}{t})$. The seminal work by Nesterov in [7] develops a simple but effective algorithm, which enables faster convergence for optimization of convex functions

Algorithm 1: Nesterov's Accelerated Gradient Descent Algorithm.

```
Input: The Lipschitz constant $L$ of the function $f(x)$.
Initialize: Given an arbitrary guess $x_0 \in \mathbb{R}^m$, set
$y_0 = x_0$, $\lambda_0 = 0$, and $t = 0$.
repeat
    $\lambda_{t+1} = \frac{1+\sqrt{1+4\lambda_t^2}}{2}$;
    $\gamma_t = \frac{1-\lambda_t}{\lambda_{t+1}}$;
    $y_{t+1} = x_t - \frac{1}{L}\nabla_f(x_t)$;
    $x_{t+1} = (1-\gamma_t)y_{t+1} + \gamma_t y_t$;
until converged;
Output: $y_{t+1}$.
```

with Lipschitz first-order derivatives. The so-called Nesterov's accelerated gradient descent algorithm, as in Algorithm 1, attains a convergence rate of $O(\frac{1}{t^2})$. Recent work in [5] has adapted the Nesterov's accelerated gradient descent algorithm to stochastic convex optimization and online learning, named the Stochastic Accelerated GradiEnt (SAGE) algorithm. Note, that the cost functions studied in [5] are in the form of expected value of some loss function, while the MSPBE function $J(\theta)$ is essentially a quadratic term of some expected values.

III. Accelerated GTD Algorithms

In this section, we derive some important properties of the MSPBE cost function, and then adapt a stochastic online learning scheme to derive an accelerated TDC algorithm.

A. Properties of the MSPBE Function

First of all, we represent the MSPBE function as

$$J\colon \mathbb{R}^k \to \mathbb{R}, \qquad \begin{aligned} J(\theta) :=& \tfrac{1}{2}\big\|V_\theta - \Pi\mathcal{T}V_\theta\big\|_D^2 \\ =& \tfrac{1}{2}\big\|\Pi(V_\theta - \mathcal{T}V_\theta)\big\|_D^2 \\ =& \tfrac{1}{2}(V_\theta - \mathcal{T}V_\theta)^\top \Pi^\top D \Pi(V_\theta - \mathcal{T}V_\theta) \\ =& \tfrac{1}{2}\big\|\widehat{\Pi}\sqrt{D}(V_\theta - \mathcal{T}V_\theta)\big\|_2^2, \end{aligned} \tag{15}$$

where

$$\widehat{\Pi} := \sqrt{D}\Phi^\top\big(\Phi D \Phi^\top\big)^{-1}\Phi\sqrt{D} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|} \tag{16}$$

is an orthogonal projector onto the column span of $\sqrt{D}\Phi^\top$. It is easy to see that the MSPBE function is quadratic, hence convex, in θ. Recall $V_\theta = \Phi^\top\theta$ and $\mathcal{T}V_\theta = R + \gamma P \Phi^\top\theta$ with $R \in \mathbb{R}^{|\mathcal{S}|}$ being the reward vector, we compute the Hessian $\mathcal{H}_J(\theta)$ of $J(\theta)$ directly as

$$\mathcal{H}_J(\theta) = \Phi(I - \gamma P)^\top\sqrt{D}\widehat{\Pi}\sqrt{D}(I-\gamma P)\Phi^\top. \tag{17}$$

Lemma 1. *Assume that the feature matrix $\Phi \in \mathbb{R}^{k\times|\mathcal{S}|}$ is full rank, and that the Markov chain defined by the transition matrix P is aperiodic and irreducible. Then the MSPBE function $J(\theta)$, as defined in Eq. (8), is strongly convex.*

Proof: As the transition matrix P defines an aperiodic and irreducible Markov chain, it follows directly that the matrix $(I-\gamma P)$ is full rank, cf. [12]. Then the result follows from the computation of the Hessian as a product of full rank matrices Eq. (17). ■

Remark 1. *Although the MSPBE function $J(\theta)$ is easily seen to be strongly convex, the smallest eigenvalue of the Hessian $\mathcal{H}_J(\theta)$ is unbounded from below, for an arbitrary feature matrix Φ.*

Lemma 2. *Let denote by $\Phi = [\phi_1, \ldots, \phi_{|\mathcal{S}|}] \in \mathbb{R}^{k\times|\mathcal{S}|}$ the feature matrix. Then the gradient of the MSPBE function $J(\theta)$ is L-Lipschitz with*

$$L = (1+\gamma)^2 \max_j \|\phi_j\|_2^2. \tag{18}$$

Proof: As the MSPBE function $J(\theta)$ is strongly convex, the gradient of $J(\theta)$ is L-Lipschitz, if the inequality

$$\theta^\top\mathcal{H}_J(\theta)\theta \le L\|\theta\|_2^2 \tag{19}$$

holds for all $\theta \in \mathbb{R}^k$. Recall the Hessian as computed in Eq. (17), we express $\theta^\top\mathcal{H}_J(\theta)\theta$ as

$$\begin{aligned} \theta^\top\mathcal{H}_J(\theta)\theta &= \theta^\top\Phi(I-\gamma P)^\top\sqrt{D}\widehat{\Pi}\sqrt{D}(I-\gamma P)\Phi^\top\theta \\ &= \big\|\widehat{\Pi}\sqrt{D}(I-\gamma P)\Phi^\top\theta\big\|_2^2 \\ &= \big\|\sqrt{D}\Phi^\top\theta\big\|_2^2 - 2\gamma\theta^\top\Phi D P \Phi^\top\theta + \\ &\qquad \gamma^2\big\|\widehat{\Pi}\sqrt{D}P\Phi^\top\theta\big\|_2^2 \\ &\le \big\|\sqrt{D}\Phi^\top\theta\big\|_2^2 - 2\gamma\theta^\top\Phi D P \Phi^\top\theta + \\ &\qquad \gamma^2\big\|\sqrt{D}P\Phi^\top\theta\big\|_2^2 \\ &= \big\|\sqrt{D}(I-\gamma P)\Phi^\top\theta\big\|_2^2, \end{aligned} \tag{20}$$

where the inequality follows from the fact that $\widehat{\Pi}$ is an orthogonal projector, i.e.

$$\big\|\widehat{\Pi}\sqrt{D}P\Phi^\top\theta\big\|_2^2 \le \big\|\sqrt{D}P\Phi^\top\theta\big\|_2^2. \tag{21}$$

By the triangle inequality, we have

$$\begin{aligned} \theta^\top\mathcal{H}_J(\theta)\theta &\le \Big(\big\|\sqrt{D}\Phi^\top\theta\big\|_2 + \gamma\big\|\sqrt{D}P\Phi^\top\theta\big\|_2\Big)^2 \\ &\le \Big(\big\|\sqrt{D}\Phi^\top\theta\big\|_2 + \gamma\big\|\sqrt{D}\Phi^\top\theta\big\|_2\Big)^2 \\ &= (1+\gamma)^2\,\big\|\sqrt{D}\Phi^\top\theta\big\|_2^2, \end{aligned} \tag{22}$$

where the second inequality follows from Lemma 9.2.1 in [2], i.e. $\|Pz\|_D \le \|z\|_D$ for all $z \in \mathbb{C}^{|\mathcal{S}|}$. Then, we compute

$$\begin{aligned} \big\|\sqrt{D}\Phi^\top\theta\big\|_2^2 = \big\|\Phi^\top\theta\big\|_D^2 &= \sum_{j=1}^{|\mathcal{S}|} d_j(\phi_j^\top\theta)^2 \\ &\le \sum_{j=1}^{|\mathcal{S}|} d_j\|\phi_j\|_2^2\|\theta\|_2^2 \\ &\le (\max_j \|\phi_j\|_2^2)\|\theta\|_2^2. \end{aligned} \tag{23}$$

Finally, we get

$$\theta^\top\mathcal{H}_J(\theta)\theta \le (1+\gamma)^2\,\big(\max_j\|\phi_j\|_2^2\big)\|\theta\|_2^2. \tag{24}$$

Thus, the result follows. ■

B. An Accelerated TDC Learning Algorithm

Following the result from the previous subsection, we can directly adopt the Nesterov's accelerated gradient descent algorithm, as presented in Algorithm 1, to minimize the MSPBE function $J(\theta)$. Surely, efficient evaluation of the actual gradient, shown in Eq. (9), is still a computational challenge for a direct implementation. Fortunately, such an issue has been successfully addressed by using a quasi-stationary estimate strategy, which leads to the promising GTD learning algorithms, cf. [10], [11]. In what follows, we present a stochastic accelerated gradient descent algorithm in combination with the GTD updates. Specifically, in Algorithm 2, only the accelerated TDC algorithm is presented. An accelerated version of the GTD2 algorithm can be developed in the same fashion.

As pointed out in Remark 1, the strong convexity constant of the MSPBE function is not bounded from below in general. Nevertheless, for a given feature matrix Φ a fixed strong convexity constant guarantees its convergence property as

Algorithm 2: Accelerated TDC Algorithm

```
Input: Parameters $L_0 > L$ and $\nu \in [0, 1]$ for the
       optimization, sequence of parameters $\beta_t$ for the
       computation of the gradient in TDC, sequence
       of state transitions and rewards $(\phi_t, r_t, \phi'_t)$.
Initialize: Given an arbitrary guess $y_0 \in \mathbb{R}^k$, set
$z_0 = y_0$, $w_0 = 0$, and $t = 1$.
repeat
    Update $L_t = (\nu\sqrt{t-1} + 1)L_0$;
    Compute $\theta_t = (1-\nu)y_{t-1} + \nu z_{t-1}$;
    Draw a sample $(\phi_t, r_t, \phi'_t)$;
    Compute a stochastic estimate of the TDC gradient:
        $\delta_t = r_t + \gamma\phi_t'^{\top}\theta_t - \phi_t^{\top}\theta_t$;
        $g_t = -\delta_t\phi_t + \gamma(\phi_t^{\top} w_t)\phi'_t$;
        $w_{t+1} = w_t + \beta_t(\delta_t - \phi_t^{\top} w_t)\phi_t$;
    Compute the SAGE update:
        $y_t = \theta_t - \frac{1}{L_t} g_t$;
        $z_t = z_{t-1} - \frac{\nu}{L_t} g_t$;
    Set $t = t + 1$;
until converged;
Output: weight vector $\theta = y_t$.
```

stated in [5]. However, in order to implement such algorithm calculation of the smallest singular value of Φ is needed but not necessarily computationally practical. Therefore, in this work, we only adopt the online learning algorithm, namely the SAGE-based online learning algorithm (algorithm 2 in [5]), without the assumption of strong convexity. In this algorithm, there are two controlling parameters: (i) $L_0 > 0$, a constant greater than the Lipschitz constant L of the gradient, and (ii) $\nu \in [0, 1]$, which can be seen as a momentum parameter in the sense that a value near 0 gives high importance to the recent updates. Specifically, setting ν to 0 turns this algorithm into the standard gradient descent, while increasing ν will give importance to a greater number of past updates.

Incorporating the stochastic TDC update into an accelerated algorithm is straightforward but not trivial. We need to carefully arrange and interleave the updates of both algorithm variables. First we update the parameter sequence L_t according to theorem (3) in [5] and compute the current value of θ_t to be able to update the stochastic gradient estimate g_t of the TDC algorithm. Then by recalling the update for the estimate of the auxiliary sequence w_t as given in Eq. (10), we update the sequences of y_t and z_t. Note, that the initial value of L_0 is chosen to be a greater value than the Lipschitz constant, calculated in Eq. (18).

IV. Experiments

To verify the successful combination of accelerated stochastic gradient algorithms with gradient temporal difference learning, we employed seven different numerical experiments. The diverse nature of experiments ensures to test the algorithms comprehensively within simulated environments. To evaluate the performance, we compare each accelerated algorithm with its original counterpart. When we refer to the accelerated versions of the algorithms in the figures and following paragraphs, we do so with a suffix ‘a’, i.e. the accelerated versions of TDC and GTD2 are called TDCa and GTD2a respectively. For the sake of readability, in this paper, only results for GTD2, GTD2a, TDC and TDCa are depicted. Further experimental results can be found in the supplementary material.

A. Experimental Setting

We used seven different experiments to evaluate the numerical performance. The first three are random walks with three different features, as described in [11], the fourth is the Boyan chain example [3] and the fifth and sixth are two random MDPs of different size. The last benchmark is a continuous cart pole balancing task. Throughout the experiments the parameter γ and the features ϕ are selected in accordance with specific problems by using common principles.

The random walk over a 5-state chain is a discrete environment, where the two outermost states are terminal and absorbing states. Reward is 0 on all transitions except upon transitioning into the rightmost terminal state, where it is 1. The discount rate is set to $\gamma = 0.9$. Following the authors in [11], we can create three different experiments from this environment by selecting three different feature representations ϕ. For the first experiment, so called *tabular features* are used, where the feature vector has the same length as number of states and a single 1 identifies in which state we are by its position in the feature vector. The so called *inverted features* have a similar construction as the previous ones, except a zero indicates the state we are in. The other values are all chosen to be positive and such that the feature vector has unit norm (in our case this would be $\frac{1}{2}$). The third feature representation, the so called *dependent features* have length 3 and can therefore no longer perfectly represent the state space. These features are also normalized to have unit length.

The second experiment, Boyan chain, is also an episodic experiment. We use a 14 state version with 4 dimensional features and set $\gamma = 0.95$.

As the random walk and Boyan chain are episodic environments, we set the maximum episode length to 20 steps and did the learning over 600 episodes.

To evaluate the performance on bigger discrete environments, we did run the algorithms on two random MDPs. Those are generated by selecting the number of states $|\mathcal{S}|$ and actions $|\mathcal{A}|$ and then randomly sampling the state transition matrix, ensuring that each state is reachable from any other state by adding a very small constant to each state transition probability. The third parameter is the size k of the feature vector. For each state we then generate a vector of length k of which $k - 1$ entries are be sampled uniformly from $[0, 1]$ and one constant 1 as a bias term. The reward for each state transition is also determined uniformly from the interval $[0, 1]$ and the discounting factor was chosen to be $\gamma = 0.95$.

As the random MDPs are non episodic only the total number of steps has to be selected. We evaluated the algorithms on two different random MDPs, one with $|\mathcal{S}| = 30$, $|\mathcal{A}| = 4$ and $k = 8$, which we ran 6000 simulation steps on and the other with $|\mathcal{S}| = 100$, $|\mathcal{A}| = 10$ and $k = 20$ which we ran for 12000 steps.

To investigate the performance in a continuous domain, we ran the algorithms in the cart pole balance task. The agent can actuate a cart on a rail either to the left or to the right. Fixed with a joint on the cart is a pole that has to be balanced. The angle of the pole with respect to the upright position ψ as well as the position of the cart x can be measured. The state consists of the angle of the pole, the angular velocity, the position of the cart and its velocity $s_t = [\psi_t, \frac{d}{dt}\psi_t, x_t, \frac{d}{dt}x_t]^\top$. We used the version of this experiment which employs the so called *imperfect feature* representation, where the feature vector contains all elements of the state vector squared element wise. Reward is given in such a way that nonzero angles ψ are heavily penalized, as well as applying acceleration to the cart and positioning the cart away from the start position. The policy evaluated was found by linearizing the system for small angles ψ and solving with dynamic programming. For details compare [4]. We ran this experiment for 15000 steps while the performance was evaluated every 500 steps. The discount rate is set to be $\gamma = 0.95$ and the initial state is a small perturbation of the equilibrium $s_0 = [0.0001, 0, 0, 0]^\top$.

In all the above experiments, the weight vector is initialized to $\theta_0 = \mathbf{0}$.

To average out fluctuations introduced by the randomness of the environments, the results of all experiments are averaged over 10 independent runs.

1) Objective Criteria: In order to compare the algorithms with respect to the final error attained and convergence speed, we employed three performance criteria.

The first is the final error attained, after the algorithm has converged. This criterion ensures the accelerated versions to converge to similar values as the known gradient based algorithms. We determine this error value as the mean over the last third of the episodes in episodic tasks and over the last third of update steps in non-episodic tasks. For example if we run an experiment for 600 episodes, then we determine the error attained as the mean over the last 200 episodes.

The second criterion is the number of iterations needed to reach a value close to the convergence value of the original algorithms and the third criterion is the computation time needed to reach the same value. Those two criteria compare the speed of convergence and we consider the algorithm to have reached an error level lower than ϵ at the step t if for all $t' > t$ the error is lower than ϵ. The error level ϵ was determined from the original gradient based learning algorithms after they have converged.

Although, the accelerated algorithms are still $O(n)$ in complexity, we distinguish between iterations and wall-clock time for convergence speed. We do this, because for smaller problems, the constant factor of additional computation introduced can have an impact on the overall processing time. For example, for problems where sampling is arbitrarily fast, then the impact of additional processing in each update step could be significant. On the other side, if each update step is much faster compared to the sampling, then only the total number of iterations plays a role in total convergence speed.

B. Algorithm Parameters

The accelerated versions of the gradient TD algorithms introduce an additional parameter, denoted as ν.

In most situations an additional parameter is undesirable and it involves some additional tradeoff between tuning all parameters and acquiring exact results. We ran the accelerated algorithms for a range of ν and observed, that increasing the value of ν improved the performance in almost all cases. For the simple random walk and Boyan chain examples, a value between 0.6 and 0.9 had not much impact on performance and achieved approximately the same results.

For larger problems, however, the impact of ν becomes more prominent. As we can observe in the larger random MDP examples, a value closer towards 1 yielded best results, especially as the problem size grows. This makes sense as larger values for ν take a larger number of past gradients into consideration, effectively bringing the approximation closer to the true gradient. As the problems have higher dimension, the history of gradients has to be longer in order to have a good estimation for the expectation of the gradient.

Upon studying Fig. 1, a thing that sticks out is the experiment for GTDa in the 30 state random MPD. Here values of ν larger than 0.1 caused the algorithm to converge so slowly such

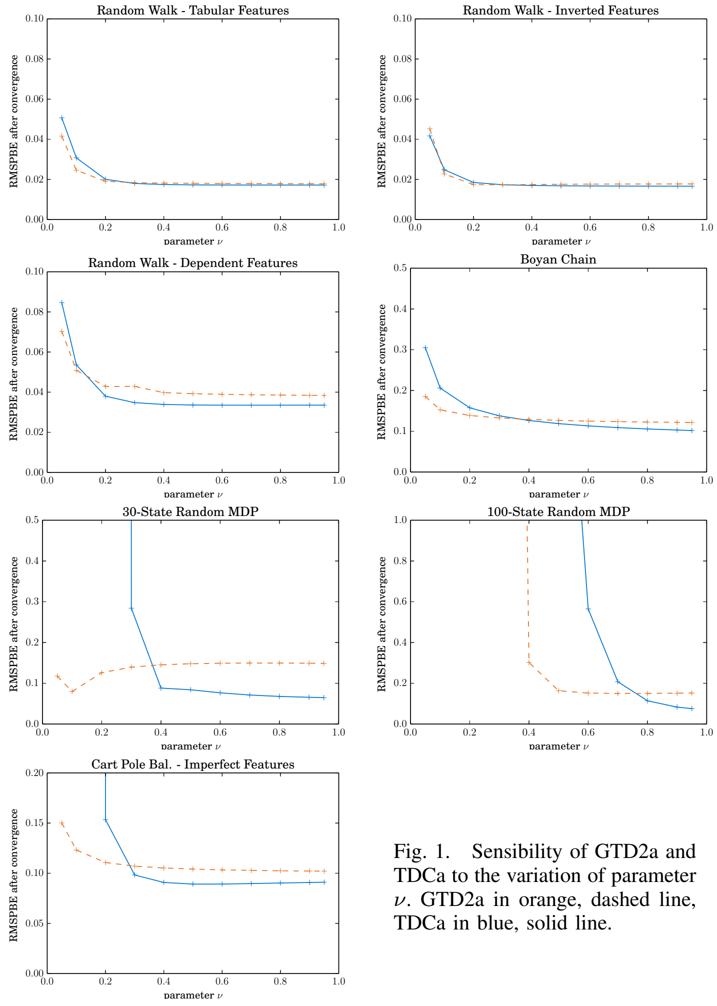
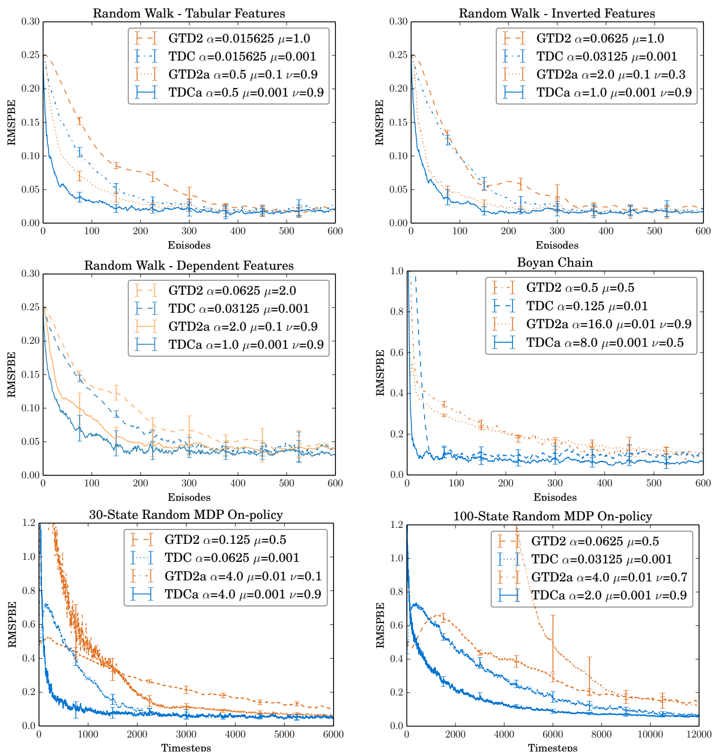


Fig. 1. Sensibility of GTD2a and TDCa to the variation of parameter ν. GTD2a in orange, dashed line, TDCa in blue, solid line.



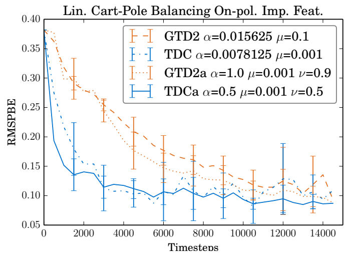
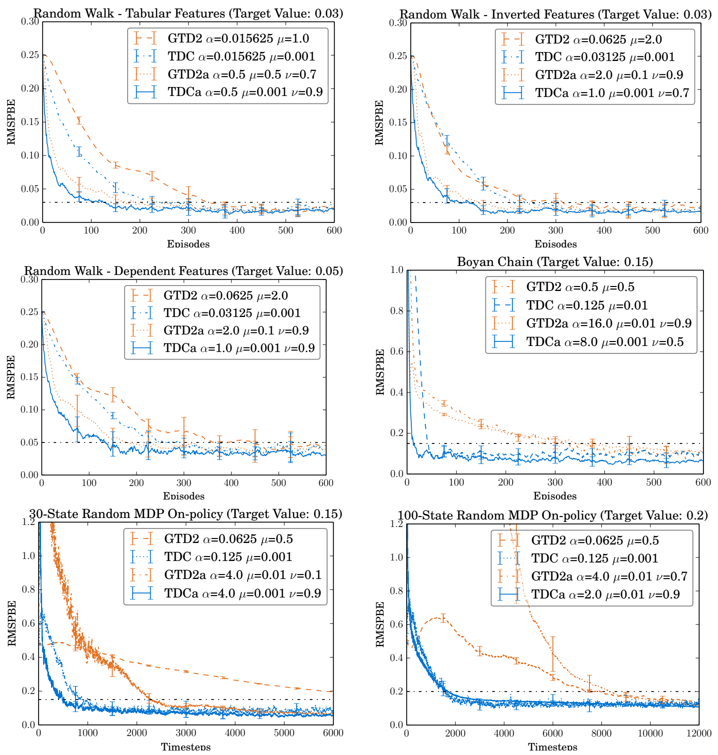


Fig. 2. Errors in the six experiments for the final error criterion. The error is plotted after each episode for the experiments 1 to 4, after each step for the experiments 5 and 6. The parameters are choosen to minimize the mean of the RMSPBE over the last third of the episodes (respectively steps for experiments 5, 6 and 7).



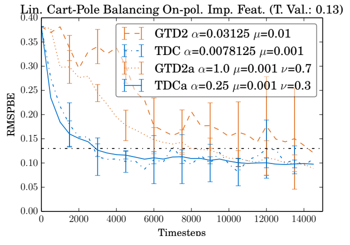


Fig. 3. Errors in the six experiments for the convergence speed in iterations. The error is plotted after each episode for the experiments 1 to 4, after each step for the experiments 5 and 6. The parameters are chosen to minimize the number of episodes, respectively steps needed to reach the plotted target value (dashed horizontal line).

that at the end of the 6000 timesteps the final error was not yet reached. The values reported for $\nu > 0.1$ are therefore not the final RMSPBE values. Nevertheless, we can still conclude, that the convergence speed for GTD2a for this experiment is the fastest for $\nu = 0.1$.

If no tuning of ν is desired, then as a rule of thumb a value close to 1 could be considered a reasonable choice. Especially, when dealing with high dimensional problems.

1) Parameter Search: To obtain good results and do a fair comparison with the existing algorithms, we did an extensive grid search over the parameter space. The parameters to be determined for each algorithms were α, the initial learning rate, where for the accelerated versions $\alpha = \frac{1}{L_0}$ is already predetermined, $\mu = \frac{\beta}{\alpha}$ the ratio between the learning rate and the update rate of the auxiliary descent in TDC and GTD2 and ν the parameter determining the length for the history of the past gradients to consider in the accelerated versions. As already mentioned above, ν could be set to a value close to 1 if there is no possibility for inclusion in the grid search.

We determined the search steps for the different parameters as follows: α was chosen to be powers of 2 between 2^{-7} and 2^5, μ was tested with the values $\mu \in (0.001, 0.01, 0.1, 0.5, 1, 2)$ and ν with the values $\nu \in (0.1, 0.3, 0.5, 0.7, 0.9)$.

C. Results Analysis

In all experiments, we measured the algorithms performance with respect to the square root of the MSPBE, referred to as RMSPBE.

Additionally, as a reference, we compared the performance of all algorithms to the traditional TD(0) algorithm. Since the results were almost exactly the same as for TDC, those are omitted in the figures to increase readability.

1) Results According to the Final Error Criterion: In almost all cases TDCa performs the best among the compared TDC, GTD2 and GTD2a. In the best case, we achieve 15% gain over the original algorithm for the random walk with dependent features. Overall we can say that qualitatively TDCa and GTD2a converge to the same values as their original counterparts. Also the fact that TDC generally outperforms GTD2 is still true for the accelerated versions.

For Experiments 5 and 6, the random MDPs, we can observe that the accelerated algorithms are performing worse in the beginning. For example for the larger random MDP, TDCa reaches a RMSPBE error value of up to 50 in the first 400 steps before going back to values smaller than 1 after 800 steps. The error to which GTD2a goes up to during the first steps is lower, but it needs longer to reach back to values smaller than 1 after a bit more than 4000 steps. The

TABLE I. SIMULATION RESULTS FOR EXP. 1: RANDOM WALK - TABULAR FEATURES, EXP. 2: RANDOM WALK - INVERTED FEATURES, EXP. 3: RANDOM WALK - DEPENDENT FEATURES, EXP. 4: BOYAN CHAIN, EXP. 5: SMALL RANDOM MDP, EXP. 6: LARGER RANDOM MDP, EXP. 7: CART POLE BALANCING

		Final error		Number of iterations		Computation time (sec)		Computation time (msec) per iter.	
	Algorithms	original	accelerated	original	accelerated	original	accelerated	original	accelerated
Exp. 1	TD(0)	0.021	-	270	-	**0.018**	-	0.067	-
	TDC	0.021	**0.017**	278	**128**	0.047	0.035	0.169	0.273
	GTD2	0.020	0.018	340	186	0.045	0.045	0.132	0.242
Exp. 2	TD(0)	0.019	-	249	-	**0.018**	-	0.072	-
	TDC	0.019	**0.017**	249	**120**	0.044	0.034	0.177	0.283
	GTD2	0.021	**0.017**	314	138	0.043	0.035	0.137	0.254
Exp. 3	TD(0)	0.040	-	291	-	**0.012**	-	0.041	-
	TDC	0.040	**0.033**	291	**134**	0.028	0.021	0.096	0.157
	GTD2	0.046	0.038	448	178	0.037	0.025	0.083	0.140
Exp. 4	TD(0)	**0.033**	-	23	-	**0.004**	-	0.174	-
	TDC	0.034	0.043	41	**14**	0.019	0.014	0.463	1.000
	GTD2	0.099	0.122	326	344	0.117	0.235	0.359	0.683
Exp. 5	TD(0)	0.059	-	892	-	**0.017**	-	0.019	-
	TDC	0.055	**0.051**	891	**589**	0.047	0.049	0.053	0.083
	GTD2	0.125	0.074	4239	2401	0.174	0.178	0.041	0.074
Exp. 6	TD(0)	0.079	-	2098	-	**0.021**	-	0.010	-
	TDC	0.081	**0.060**	1867	**1819**	0.053	0.082	0.028	0.045
	GTD2	0.162	0.148	7764	7777	0.171	0.312	0.022	0.040
Exp. 7	TD(0)	0.107	-	7000	-	**0.116**	-	0.017	-
	TDC	0.106	**0.089**	7000	**2500**	0.777	0.458	0.111	0.183
	GTD2	0.117	0.102	14000	7500	1.188	1.216	0.085	0.162

original versions TDC and GTD2 show similar behavior in the beginning of the experiments, the maximum error they go up to is however smaller.

The overview in Table I suggests almost consistent improvement for the accelerated versions of the algorithms, except for Experiment 4, where the classic TD(0) algorithm achieves best results.

The parameters found for the experiments were chosen according to the first performance criterion to minimize the average error attained. Note that nevertheless especially TDCa still outperforms the other algorithms in convergence speed. This brings us to the discussion of the steps needed to reach below a certain error threshold.

2) Analysis of the Convergence Speed Results: In order to compare the algorithms with respect to convergence speed, we ran a second parameter search with the second performance criterion of the steps needed to reach below a certain threshold.

In general TDCa attains the chosen target value (which might be different for each experiment) in less iterations than TDC, GTD2 and GTD2a. The improvement over TDC is slightly over 50% for the random walks and the cart pole balancing task and around 40% for the Boyan chain and the small random MDP. For the larger random MDP, however, the performance with respect to the number of iterations is almost the same as the original TDC version.

Similarly, GTD2a performs better than its original version in the random walks, the cart pole balancing task and the smaller random MDP. It fails to outperform GTD2 for the Boyan chain and the larger random MDP.

It is well known that the original GTD family has the computational complexity being linear in terms of the size of features k. The last two columns of Table I calculate the average computation time per iteration for all algorithms in comparison, via dividing the total runtime spent in algorithm updates by the number of iterations till convergence. It is evident, as well as without surprise, that the computation time per iteration for all accelerated algorithms is *less than double* of the time needed by their original counterparts. This observation suggests that accelerated GTD algorithms share the same property of having linear computational complexity with respect to the size of features. In the majority of the experiments, the faster convergence with respect to the number of iterations compensates for this difference and allows for the accelerated algorithms to be faster or equally fast than their original counterparts. If we compare with Table I then we can conclude, however, that due to the simplicity of its update TD(0) remains the fastest among all considered algorithms in our setup.

If we look at the results again more closely and compare the algorithms to each other, we can see, that TDCa still performs very good. In four out of seven experiments it reaches the threshold value 20% - 30% faster than TDC with respect to the total computation time needed. GTD2a can outperform GTD2 in Exp. 2 and Exp. 3 where it needs around 25% less total time, but falls behind on the other experiments.

Generally, we can conclude, that the accelerated versions of the gradient algorithms outperform their counterparts. Exceptions are mainly with respect to computational time, which is however very dependent on the problem size and number of iterations considered. Overall TDCa shows the best performance amongst the gradient based algorithms.

V. CONCLUSION

In this work, we employ the Nesterov's accelerated gradient descent algorithm to minimize the MSPBE cost function. With an identification of the Lipschitz constant of the gradient of the MSPBE cost function, we propose a stochastic accelerated GTD algorithm scheme as an extension of the well known GTD algorithms. Our numerical experiments confirm that the proposed accelerated GTD algorithms only require a slight extra computation per iteration. Meanwhile, they enjoy a higher convergence rate in terms of the number of iterations, and achieve better performance in terms of smaller MSPBE function values. The promising capacity in solving the policy evaluation problem suggests their potential benefits in control scenarios.

Acknowledgment

The authors would like to thank Christoph Dann for making his implementation of several reinforcement algorithms, `tdlearn`, publicly available. This work has been supported by IGSSE - International Graduate School of Science and Engineering, Technische Universität München.

References

[1] L. Baird: Residual algorithms: Reinforcement learning with function approximation. Proceedings of the 12^{th} International Conference on Machine Learning, pp. 30-37. (1995)

[2] D. P. Bertsekas, V. S. Borkar, A. Nedic: Improved temporal difference methods with linear function approximation. Learning and Approximate Dynamic Programming, pp. 231–255. (2004)

[3] J. A. Boyan: Least-squares temporal difference learning. Proceedings of the 16^{th} International Conference on Machine Learning, pp. 49-56. (1999)

[4] C. Dann, G. Neumann, J. Peters: Policy Evaluation with Temporal Differences: A Survey and Comparison. Journal of Machine Learning Research, vol. 15, pp. 809–883. (2014)

[5] C. Hu, J. T. Kwok, W. Pan: Accelerated Gradient Methods for Stochastic Optimization and Online Learning. Advances in Neural Information Processing Systems 22, pp. 781–789. (2009)

[6] L. Li: A worst-case comparison between temporal difference and residual gradient with linear function approximation. Proceedings of the 25^{th} International Conference on Machine Learning, pp. 560–567. (2008)

[7] Y. Nesterov: A method of solving a convex programming problem with convergence rate O($\frac{1}{k^2}$). In Soviet Mathematics Doklady, vol. 27, no. 2, pp. 372–376. (1983)

[8] I. Sutskever, J. Martens, G. Dahl, G. Hinton: On the importance of initialization and momentum in deep learning. Proceedings of the 30^{th} International Conference on Machine Learning, pp. 1139–1147. (2013)

[9] R. S. Sutton: Learning to predict by the methods of temporal differences. Machine learning, vol. 3, no. 1, pp. 9–44. (1988)

[10] R. S. Sutton, Cs. Szepesvári, 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, pp. 1609–1616. (2008)

[11] R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, Cs. Szepesvári: Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation. In Proceedings of the 26^{th}International Conference on Machine Learning (ICML), pp. 993–1000. (2009)

[12] J. N. Tsitsiklis, B. Van Roy: An analysis of temporal-difference learning with function approximation. IEEE Transactions on Automatic Control, vol. 42, no. 5, pp. 674–690. (1997)