Algorithms for Fast Gradient Temporal Difference Learning

Christoph Dann
Autonomous Learning Systems Seminar
Department of Computer Science
TU Darmstadt
Darmstadt, Germany
cdann@cdann.de

Abstract

Temporal difference learning is one of the oldest and most used techniques in reinforcement learning to estimate value functions. Many modifications and extension of the classical TD methods have been proposed. Recent examples are TDC and GTD(2) ([Sutton et al., 2009b]), the first approaches that are as fast as classical TD and have proven convergence for linear function approximation in on- and off-policy cases. This paper introduces these methods to novices of TD learning by presenting the important concepts of the new algorithms. Moreover the methods are compared against each other and alternative approaches both theoretically and empirically. Eventually, experimental results give rise to question the practical relevance of convergence guarantees for off-policy prediction by TDC and GTD(2).

1 Introduction

In general, the goal of reinforcement learning is to learn by maximizing some notion of reward obtained for taken actions. A key concept in many algorithms and theorems such as temporal difference learning or Monte Carlo search [Sutton and Barto, 1998] are *value functions*. They specify for each state of the learning environment the accumulated amount of reward an agent is excepted to achieve in the future. A reliable prediction of value function for a given system is a highly achievable goal in reinforcement learning, as it may be of help for many applications.

Sometimes the value function itself is the final goal and provides insightful information such as failure probabilities of large telecommunication networks [Frank et al., 2008], taxi-out times at big airports [Balakrishna et al., 2010] or important board configurations in Go [Silver and Sutton, 2007]. Yet, most frequently value functions are an intermediate step to obtain an optimal policy, e.g. for applications in robot soccer [Riedmiller and Gabel, 2007].

One of the major challenges of machine learning is the complexity of real world environments. In fact, most applications of reinforcement learning have to deal with continuous action and state spaces (see examples in [Lucian Busoniu, Robert Babuska, Bart De Schutter, 2010]). Even artificial settings such as the games of chess, Go [Silver and Sutton, 2007] or gammon have discrete, but extremely large, descriptions of states and actions. This issue motivated the recent development of new algorithms for efficient value function prediction in high-dimensional environments [Sutton et al., 2009b].

The purpose of this seminar paper is to give an overview of those new methods and discuss them in the context of existing approaches. It is assumed that the reader has basic knowledge of machine learning and reinforcement learning in particular. To understand this survey, the reader should have a raw idea of Markov decision processes and temporal difference learning. A short

introduction to these concepts in the broader context of artificial intelligence can be found in [Russell and Norvig, 2003], while a thorough discussion is given in [Sutton and Barto, 1998].

This paper focuses only on value function predictors that use temporal differences and linear function approximations for learning. In particular the recently proposed gradient methods GTD, GTD2 and TDC [Maei, 2011] are covered together with TD(λ) [Sutton and Barto, 1998], Residual gradient [Baird, 1995] and LSTD(λ) [Boyan, 2002] as most important alternatives. All methods have their own characteristics, but are often motivated by the same underlying concepts. Therefore, this paper presents briefly the common *design patterns*. These patterns provide the reader with a good overview of connections and differences between algorithms. In addition to that, most important properties such as convergence guarantees and speed are compared based on existing theoretical and empirical results. All algorithms have been reimplemented to allow own experiments and to provide the reader with source code as reference.

The remainder of this paper is structured as follows: Section 2 introduces the notion and notation of Markov decision processes and value functions. This is followed by the presentation of design patterns for fast linear gradient methods are presented in Sec. 3. Those are (1) the approximation of value functions by a linear function, (2) the principle of stochastic gradient descent, (3) eligibility traces, and (4) importance weighting for off-policy prediction. After that gradient algorithms are compared against each other and LSTD(λ) in Section 4. This includes both theoretical properties and results of empirical experiments. The paper is concluded in Sec. 5 with recommendations for further reading material.

2 Markov Decision Processes

The learning environment is modeled by a *Markov Decision Process* (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R)$, an extended version of standard Markovian stochastic processes. At each discrete timestep $t = 0 \dots T$ the system is in a state $S_t \in S$ and the agent choses an action $A_t \in \mathcal{A}$. The state for the next time step is then sampled from the transition model $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, i.e. $P(s_{t+1}|s_0)$ is the probability (density) for the transition from s_t to s_{t+1} when the agent performed a_t (lower case letters are denoted to realizations of random variables). For each transition the agent receives a reward $R_t = R(S_t, A_t)$ specified by the deterministic *reward function* $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. While there may be an infinite number or continuous states and actions we assume for notational simplicity, that $\mathcal{S}$ and $\mathcal{A}$ are finite sets (finite discrete MDP).

Most learning problems are modeled either as an *episodic* or an *ergodic* MDP. Episodic MDPs contain terminal states $\mathcal{T} \subseteq \mathcal{S}$ with $p(s|a, s) = 1, \forall a \in \mathcal{A}, s \in \mathcal{T}$. If a system enters such a state, it will never leave it. A realization of the process until the first occurrence of a terminal state is called *episode*. The length of an episode is denoted by T.

Problems that could possibly run infinitely long are represented as *ergodic* MDPs. These processes have to be irreducible, aperiodic and positive recurrent. Put in highly simplified terms, ergodic means every state can be reached from all others within a finite amount of time. For details and exact definitions see [Rosenblatt, 1971]. If these assumptions hold, there exists a *limiting distribution* d over $\mathcal{S}$ with $d(s) = \lim_{t\to\infty} P(S_{t+1} = s|S_t, A_t)$. Practically, it is the relative amount of time the process is a specific state, when the process runs infinitely long. Formally, episodic MDPs do need not have unique limiting distributions, yet a distribution defined as $d(s) = \mathbb{E}\big[\sum_{t=0}^{T} 1_{\{S_t=s\}}\big]$ has the same intuition for episodes.

The behavior of the learning agent within the environment, i.e. the strategy of selecting an action given the current state, is denoted by a *policy* $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The action A_t an agent performs in a state S_t is generated by sampling from the probability distribution $A_t \sim \pi(\cdot\,|S_t)$. Often an agent follows *deterministic policies*, which chose always the same action given the current state. Then a function $\pi : \mathcal{S} \to \mathcal{A}$ is used as a model.

The goal of reinforcement learning is to find a policy that maximizes the *expected (total discounted) reward*

$$\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t\right], \qquad \text{(Discounted Reward)}$$

where $\gamma \in (0, 1]$ is the discount factor. A small discount factor puts more weight on earlier rewards than on rewards gained later. While T is a finite for episodic task, we consider the expected reward in the limit $T \to \infty$ for ergodic MDPs.

2.1 Value Functions

The need to compare and evaluate policies motivates the concept of *value functions*. A value function $V^\pi : \mathcal{S} \to \mathbb{R}$ assigns each state the expected reward, given the process started in that state and the actions are chosen by π:

$$V^\pi(s) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t R_t \mid S_0 = s\Big]. \qquad \text{(Value Function)}$$

By looking at two succeeding timesteps t and $t+1$, we can see, that for arbitrary MDPs $\mathcal{M}$, discount factors γ and policies π, it holds that

$$V^\pi(s_t) = \mathbb{E}\Big[r(s_t, a_t) + \gamma V^\pi(s_{t+1})\Big]. \qquad \text{(Bellman Equation)}$$

This insight named the *Bellman Equation* is the basis of temporal difference learning approaches. We introduce the following matrix notations for value functions to reformulate the (Bellman Equation) in a concise way: The value function V^π can be written as a $n := |\mathcal{S}|$ dimensional vector $\mathbf{V}^\pi$, which contains $V(s^i)$ at position i for some fixed order $s^1, s^2, \dots s^n$ of the state space. Using an additional vector $\mathbf{R} \in \mathbb{R}^n$, $R_i = \mathbb{E}\big[r(s^i, a)\big]$ the (Bellman Equation) can be rewritten as

$$\mathbf{V}^\pi = \mathbf{R} + \gamma \mathbf{P} \mathbf{V}^\pi =: T^\pi \mathbf{V}^\pi, \qquad \text{(Bellman Operator)}$$

where $\mathbf{P} \in \mathbb{R}^{n \times n}$ is a matrix with elements $P_{ij} = \mathbb{E}\big[P(s^i | s_j, a)\big]$. In this representation it becomes clear that the Bellman Equation describes an affine transformation $T^\pi : \mathbb{R}^n \to \mathbb{R}^n$ of $\mathbf{V}^\pi$. In the following we omit the policy superscripts for notational simplicity in unambiguous cases.

The algorithms discussed in this paper address the problem of *value function prediction*: Given data $(s_t, a_t, r_t; t = 1 \dots T)$ sampled from an MDP and a policy, the goal is to find the underlying value function.

Since this problem cannot be solved in closed form for almost all non-trivial cases, iterative estimation techniques are used. In order to evaluate the quality of such estimates, we need to define a notion of similarity of two value functions V_1 and V_2. A natural distance for that is the squared distance weighted by the limiting distribution of the underlying MDP

$$\|\mathbf{V_1} - \mathbf{V_2}\|_d^2 \equiv \sum_{i=1}^{n} d(s^i)[V_1(s^i) - V_2(s^i)]^2 = [\mathbf{V}_1 - \mathbf{V}_2]^T \mathbf{D} [\mathbf{V}_1 - \mathbf{V}_2], \qquad (1)$$

where $\mathbf{D} = \mathrm{diag}[d(s^1), d(s^2), \dots, d(s^n)]$ is the weight matrix with the entries of d on the diagonal.

3 Algorithm Design Patterns for Fast TD Learning

The following section gives an introduction to the most common design patterns for development and adaption of temporal difference algorithms. First, linear function approximation is presented as the standard way to deal with extremely large state spaces. Second, the principle of stochastic gradient descent is explained as the basis for TDC, GTD(2) and the residual gradient approach. Furthermore, eligibility traces are discussed and importance weighting, which extends the algorithms for off-policy value function prediction.

3.1 Linear Value Function Approximation

The number of states is huge in most learning environments. For example, representing the board configuration of checkers in a straight forward way (32 places, 5 possibilities each: empty, white men, white king, black men, black king) results in $5^{32} = 23283064365386962890625$ states. If we wanted to store the value function for them explicitly would require more than $2^{13} GB$ of memory.

So, a compact representation scheme with good generalization properties is necessary. The standard solution is a linear function approximation with a shared parameter vector $\theta \in \mathbb{R}^m$ and a feature vector $\phi(x)$ dependent on the state x given by

$$V(x) \approx V_\theta(x) = \theta^T \phi(x). \tag{2}$$

Critical for the quality of the value function approximation is the feature function $\phi : \mathcal{S} \to \mathbb{R}^m$. It can either be hand-crafted and incorporate domain knowledge or a general local weighting function. Common choices are radial basis functions and sparse coding techniques such as CMACs (e.g., [Sutton and Barto, 1998]). Even though such advanced discretization methods are a concise representation of continuous spaces, curse of dimensionality still renders learning in these cases problematic.

The main purpose of ϕ is the reduction of dimensionality from n to m, which comes with a loss of precision of state representation. So, it may not be possible to assign the values $V_\theta(s^1)$ and $V_\theta(s^2)$ of two arbitrary states independently given a fixed ϕ. More precisely this is always the case if $\phi(s^1)$ and $\phi(s^2)$ are linearly dependent. We denote $\mathcal{H}_\phi$ to the set of all functions, we can represent exactly for a given feature function,

$$\mathcal{H}_\phi = \{V : \exists \theta \, \forall s \, V(s) = \theta^T \phi(s)\} \subset \{V : \mathcal{S} \to \mathbb{R}\}. \tag{3}$$

3.2 Stochastic Gradient Descent (SGD)

The technique of stochastic gradient descent can be used to motivate the classical temporal difference algorithm, the residual gradient approach as well as the recent TD methods, that minimize the projected fixpoint equation.

Temporal Differences. In order to approximate the true value function V^π well, we try to make the approximation V_θ as close as possible. So, the goal is to find parameters θ, that minimize the *Mean Squared Error*

$$\text{MSE}(\theta) \equiv \|\mathbf{V}_\theta - \mathbf{V}^\pi\|_d^2. \tag{MSE}$$

Finding the global minimum directly by setting the gradient to 0 is infeasible due to the size of the state space. Instead, a gradient descent approach has to be used to iteratively update θ

$$\theta' = \theta + \alpha \nabla \, \text{MSE} = \theta + \alpha \left(2 \sum_{i=1}^{n} d(s^i)[V_\theta(s^i) - V^\pi(s^i)]\phi(s^i) \right) \tag{4}$$

where θ is the old and θ' the updated estimate and α scales the length of the step. Still, the sum over all states is not manageable, so we approximate it by just taking the direction of a single term

$$\theta' = \theta + \alpha[V_\theta(s^i) - V^\pi(s^i)]\phi(s^i) \tag{5}$$

This technique is called *stochastic gradient descent* and is visualized in Fig 1. As depicted the steps may have a different length and direction. So, the followed path is noisier and more steps may be necessary to reach a local optimum.

For prediction the value function, we are provided with observed realizations of the underlying MDP $(s_t, a_t, r_t; t = 1 \dots T)$, so we can use each timestep as a sample for stochastic gradient descent. However, we do not know the value of $V^\pi(s_t)$. The main idea of *Temporal Difference Learning* is to approximate V_{θ_t} by applying the (Bellman Equation) on the current estimate V_θ

$$V^\pi(s_t) \approx r(s_t, a_t) + \gamma V_\theta(s_{t+1}) \tag{6}$$

which gives the update rule of *Linear TD Learning* [Sutton and Barto, 1998]

$$\theta_{t+1} = \theta_t + \alpha[r(s_t, a_t) + \gamma V_{\theta_t}(s_{t+1}) - V_{\theta_t}(s_t)]\phi(s_t) \equiv \theta_t + \alpha \delta_t \phi(s_t) \tag{7}$$

TD Learning compares the prediction for the current state s_t to the one for the next state s_{t+1}, the temporal difference, which is also known as *TD Error*

$$\delta_t \equiv r(s_t, a_t) + \gamma V_{\theta_t}(s_{t+1}) - V_{\theta_t}(s_t) = r(s_t, a_t) + \theta_t^T(\gamma \phi_{t+1} - \phi_t). \tag{TD Error}$$

In the following the shorthand notation $\phi_t \equiv \phi(s_t)$ will be used. The prediction according to the current estimate V_{θ_t} is used to improve the current estimate, which is called *bootstrapping*.

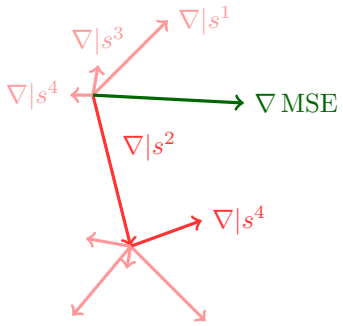

Figure 1: Stochastic Gradient Descent: Instead of following the true gradient ∇ MSE, a single summand $\nabla|s^i$ is used

Residual Gradient. Alternatively, the idea of temporal differences can be incorporated directly in the objective function, by minimizing a different error. The so motivated *Mean Squared Bellman Error (MSBE)* measures the difference of both sides of the (Bellman Operator):

$$\text{MSBE}(\theta) \equiv \|\mathbf{V}_\theta - T\mathbf{V}_\theta\|_d^2. \tag{MSBE}$$

Minimizing the MSBE with stochastic gradient descent is called *Residual Gradient* (RG) method [Baird, 1995]. Its update rule is shown in a more general form in Algorithm 5. The meaning of variable ρ_t will be explained in detail in Section 3.4 and can be set to $\rho_t = 1$ for now. While the residual gradient approach always converges to a fixpoint of the (MSBE) for deterministic MDPs, two independent successor states are necessary for guaranteed convergence in the stochastic case. This is called the double sampling problem and a major caveat of RG methods. Details can be found in Section 6 of [Baird, 1995]. It is shown in [Maei, 2011], that if only one sample is used, then RG approaches converge to a fixed point of the *Mean Squared TD-Error*

$$\text{MSTDE}(\theta) \equiv \mathbb{E}[\delta_t^2]. \tag{MSTDE}$$

Projected Fixpoint. The limiting parameters of linear TD Learning – often referred to as "TD fixpoint" – are in general not a fixed point of the (MSE) due to bootstrapping. In addition, they are not a fixed point of the (MSBE), either. This has the following reason: the dynamics of the MDP is independent of the parametrized function space $\mathcal{H}_\phi$, so it may occur that $T\mathbf{V}_\theta \notin \mathcal{H}_\theta$. The next value function estimate has to be mapped back to $\mathcal{H}_\phi$ by a projection Π. In Section 2.5 of [Maei, 2011], it is shown that the solution of linear TD satisfies $V_\theta = \Pi T V_\theta$ with

$$\Pi\mathbf{V} \equiv \min_{V_\theta \in \mathcal{H}_\phi} \|\mathbf{V}_\theta - \mathbf{V}\|_d^2 = \mathbf{\Phi}(\mathbf{\Phi}^\mathbf{T}\mathbf{D}\mathbf{\Phi})^{-\mathbf{1}}\mathbf{\Phi}^\mathbf{T}\mathbf{D} \tag{8}$$

being the projection to $\mathcal{H}_\theta$ with smallest distance. As it describes a least squares problem weighted by d, its solution can be written directly as weighted pseudo-inverse using the weight matrix $\mathbf{D}$ and feature matrix $\mathbf{\Phi} = [\phi(s^1)^T, \phi(s^1)^T, \dots \phi(s^1)^T]^T$. Figure 2 visualizes the relationship between the operators. This insight motivates to minimize a different objective function instead of the (MSBE), the *Mean Squared Projected Bellman Error*

$$\text{MSPBE}(\theta) \equiv \|\mathbf{V}_\theta - \Pi T\mathbf{V}_\theta\|_d^2. \tag{MSPBE}$$

In [Sutton et al., 2009b], it has been shown that the (MSPBE) can be rewritten as

$$\text{MSPBE}(\theta) = \mathbb{E}[\delta_t\phi_t]^T\mathbb{E}[\phi_t\phi_t^T]^{-1}\mathbb{E}[\delta_t\phi_t]. \tag{9}$$

To derive that formulation, they make heavy use of matrix notations from Eq. (8), the (Bellman Operator) and (3). In addition to that they connect expected values and matrix notation by $\mathbb{E}[\phi_t\phi_t^T] = \mathbf{\Phi}^T\mathbf{D}\mathbf{\Phi}$ and $\mathbb{E}[\delta_t\phi_t] = \mathbf{\Phi}^T\mathbf{D}(\mathbf{T}\mathbf{V}_\theta - \mathbf{V}_\theta)$. All derivations require only basic laws of matrices and probabilities, but contain several steps. As no further insight can be gained from them, they are not repeated here. Interested readers are referred directly to Section 4 of [Sutton et al., 2009b].

The gradient of (9) can be written in the following ways:

$$\nabla\,\text{MSPBE}(\theta) = -2\mathbb{E}[(\phi_t - \gamma\phi_{t+1})\phi_t]^T\mathbb{E}[\phi_t\phi_t^T]^{-1}\mathbb{E}[\delta_t\phi_t] \tag{10}$$

$$= -2\mathbb{E}[\delta_t\phi_t] + 2\gamma\mathbb{E}[\phi_{t+1}\phi_t]^T]\mathbb{E}[\phi_t\phi_t^T]^{-1}\mathbb{E}[\delta_t\phi_t] \tag{11}$$

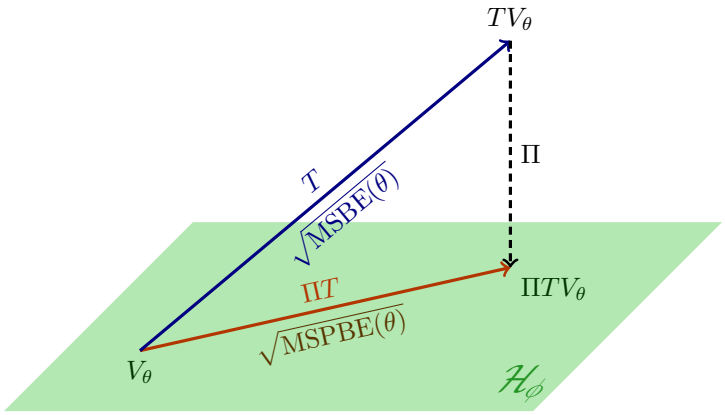

Figure 2: Projection of $T\mathbf{V}_\theta$ back in the space of parametrized functions $\mathcal{H}_\phi$

Each form gives rise to a new version of learning with temporal differences by following it in stochastic gradient descent. However, both versions are products of expected values of random variables that are not independent. So, the straight forward approach – take only one sample to compute the gradient – would not work, as the product would be biased by the covariance of the random variables. This becomes clear when we look at the general case for two random variables X, Y with realizations x,y

$$xy \underset{SGD}{\approx} \mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] - \text{Cov}[X, Y]. \tag{12}$$

This is exactly the double sampling problem of residual gradient methods. To circumvent the requirement of two independent samples, a long-term quasi-stationary estimate w of

$$\mathbb{E}[\phi_t \phi_t^T]^{-1}\mathbb{E}[\delta_t \phi_t] = (\mathbf{\Phi^T D \Phi})^{-\mathbf{1}}\mathbf{\Phi}^T\mathbf{D}(\mathbf{TV}_\theta - \mathbf{V}_\theta) \tag{13}$$

is stored. To arrive at an iterative update for w, we realize that the right side of Equation (13) has the form of the solution to the following least-squares problem

$$J(w) = [\Phi^T w - (TV_\theta - V_\theta)]^T[\Phi^T w - (TV_\theta - V_\theta)]. \tag{14}$$

Solving that problem produces estimates of w. So applying the principle of stochastic gradient descent again yields the following the update rule

$$w_{t+1} = w_t + \beta_t(\rho_t\delta_t - \phi_t^T w_t)\phi_t. \tag{15}$$

Plugging the estimate w_t into Eq.(10) allows us to rewrite the gradient with a single expectation

$$\nabla \text{MSPBE}(\theta) = -2\mathbb{E}[(\phi_t - \gamma\phi_{t+1})\phi_t]^T w_t \tag{16}$$

$$= -2\mathbb{E}[\delta_t\phi_t] + 2\gamma\mathbb{E}[\phi_{t+1}\phi_t]^T]w_t. \tag{17}$$

Optimizing the (MSPBE) by SGD with the gradient in the form of (16) yields the *GTD2 (Gradient Temporal Difference 2)* algorithm and using the form of (17) produces *TDC (Temporal Difference Learning wit Gradient Correction)*, which are both proposed in [Sutton et al., 2009b]. The complete update rules are shown in Algorithm 3 and 4. As θ and w are updated at the same time, the choice of step sizes α_t and β_t are critical for convergence and will be discussed in Sec. 4. Both methods can be understood as a nested version of stochastic gradient descent optimization.

If a parameter θ is a minimizer of (9), then the updates in Eq. (7) do not change θ anymore in the long run. So, it holds that $\mathbb{E}[\delta_t\phi_t] = 0$. Therefore, an alternative way to find θ is to minimize the updates, i.e. minimize the *norm of the expected TD update*:

$$NEU(\theta) \equiv \mathbb{E}[\delta\phi]^T E[\delta\phi]. \tag{NEU}$$

Using the exact same techniques as for the derivation of GTD2 and TDC to optimize the (NEU) gives rise to the *GTD* algorithm [Sutton et al., 2009a] shown in Algorithm 2.

Algorithm 7 Code for a Generic Episodic Learning Environment

function TDEPISODICLEARN($\lambda, \gamma, \pi_G, \pi_B$)
  $t \leftarrow 0$
  for $i = 1, ..., N$ **do** ▷ number of episodes to learn from
    $z_t = \vec{0}$
    repeat observe transition (s_t, a_t, s_{t+1}, r_t) ▷ state, action, next state, reward
      $\rho_t = \frac{\pi_G(a_t|s_t)}{\pi_B(a_t|s_t)}$
      $\delta_t = r_t + \gamma\lambda\theta_t\phi_{t+1} - \theta_t\phi_t$

    ▷ insert any update rule from Figure 3

      $t \leftarrow t + 1$
    until $s_{t+1} \in \mathcal{T}$ ▷ terminal state reached
  return θ_t

$$z_{t+1} = \rho_t(\phi_t + \lambda\gamma z_t)$$
$$\theta_{t+1} = \theta_t + \alpha_t\delta_t z_{t+1}$$

Algorithm 1: Linear TD(λ)

$$\theta_{t+1} = \theta_t + \alpha_t\rho_t(\phi_t - \gamma\phi_{t+1})\phi_t^T w_t$$
$$w_{t+1} = w_t + \beta_t\rho_t(\delta_t\phi_t - w_t)$$

Algorithm 2: GTD

$$\theta_{t+1} = \theta_t + \alpha_t\rho_t(\phi_t - \gamma\phi_{t+1})\phi_t^T w_t$$
$$w_{t+1} = w_t + \beta_t(\rho_t\delta_t - \phi_t^T w_t)\phi_t$$

Algorithm 3: GTD2

$$\theta_{t+1} = \theta_t + \alpha_t\rho_t(\delta_t\phi_t - \gamma\phi_{t+1}\phi_t^T w_t)$$
$$w_{t+1} = w_t + \beta_t(\rho_t\delta_t - \phi_t^T w_t)\phi_t$$

Algorithm 4: TDC

$$\theta_{t+1} = \theta_t + \alpha_t\rho_t\delta_t(\phi_t - \gamma\phi_{t+1})$$

Algorithm 5: Residual Gradient

$$z_t = \gamma\lambda\rho_{t-1}z_{t-1} + \phi_t$$
$$K_t = \frac{M_{t-1}z_t}{1 + (\phi_t - \gamma\rho_t\phi_{t+1})^T M_{t-1}z_t}$$
$$\theta_{t+1} = \theta_t + K_t(\rho_t r_t - (\phi_t - \gamma\rho_t\phi_{t+1})^T\theta_t)$$
$$M_t = M_{t-1} - K_t(M_{t-1}^T(\phi_t - \gamma\rho_t\phi_{t+1}))^T$$

Algorithm 6: recursive LSTD(λ)

Figure 3: Update Rules of Temporal Difference Algorithms. These updates are executed for each transition from s_t to $s_t + 1$ performing action a_t and getting the reward r_t. For the context of update rules see Algorithm 7

3.3 Eligibility Traces

The idea of the value of a state is the total expected reward, that is gained after being in that state. So, the value of s^i can be computed alternatively by observing a large number of episodes starting in s^i and taking the average reward per episode. This idea is known as *Monte-Carlo-Sampling* (MC Sampling, Ch. 5 of [Sutton and Barto, 1998]). It can be implemented as an instance of stochastic gradient descent per episode when we replace $V^\pi(s^i)$ in Eq.(5) with the reward of the current episode starting in s^i, i.e.

$$\theta_{t+1} = \theta_t + \alpha_t\left[\left(\sum_{t'=t}^{T}\gamma^{t'}r_{t'}\right) - V^\pi(s_t)\right]\phi(s_t), \tag{18}$$

where T denotes the length of the current episode. So, the main difference between TD (Eq. (TD Error)) and MC learning is the target value. While taking the prediction for the next state for temporal differences, MC techniques focus on the final reward of episode.

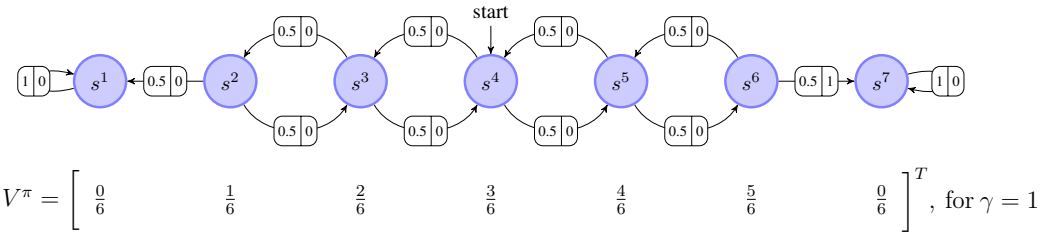

Figure 4: 7-State Random Walk MDP

This can be beneficial for convergence speed in some cases, as we will illustrate by the following example: consider the random walk MDP in Figure 4 with seven states and a single action. The first number of a transition denotes its probability, while the second is the associated reward. The episode always starts in the middle state and goes left or right with equal probability. There is no reward except when entering the right terminal state s^7. Assuming no discount, $\gamma = 1$, the true value function V^π increases as shown linearly from left to right.

For simplicity we assume tabular features $\phi(s^i) = e_i$, where e_i is a zero-vector of length m except for the i-th position being 1. Arbitrary value functions can be represented with tabular features, i.e. $\mathcal{H} = \mathcal{H}_\phi$. Assume $\theta_0 = \vec{0}$ and that we have observed two episodes. One goes straight right to s^7 and the other always transits left to s^1. Then MC learning yields the average outcome $\frac{1}{2}$ as a value for the initial state. That is already the true value. In contrast, TD learning still assigns 0 to the middle state, as the positive reward of the right end can only propagate one state per visit. This shows that MC approaches may produce good results faster and it may have benefits to look several steps ahead instead of only 1 as in classical TD.

Therefore, TD learning is extended with *eligibility traces* which allow to incorporate the idea of Monte-Carlo approaches. A parameter $\lambda \in [0, 1]$ is introduced to control the amount of look-ahead. Let $r_t^{(k)}$ denote the accumulated reward, we get within the next k timesteps (k-step lookahead reward). Instead of always considering $r_t^{(\infty)}$ as in MC learning or $r_t^{(0)}$ as in classical TD learning, we take a weighted average of lookahead rewards

$$(1 - \lambda) \sum_{k=0}^{\infty} \lambda^k r_t^{(k)} \tag{19}$$

as the target value. The parameter λ controls the exponential decay in lookahead and the normalization $(1 - \lambda)$ assures that all weights sum up to 1. So, setting $\lambda = 1$ corresponds to MC learning and $\lambda = 0$ corresponds yields TD learning.

Eligibility traces are the tool for efficiently implementing the concept of decayed rewards. Instead of considering the reward for upcoming events, the idea is to store a record of seen states and update their values given the current event. As the states are represented by their features, it is possible to store the trace of past states space efficient as

$$z_{t+1} = \phi_t + \lambda \gamma z_t. \tag{20}$$

The features of the current state are simply added to the trace and the importance of all past entries is lowered by the decay parameter λ and discount factor γ. As not only the parameters corresponding to the current features but to all weighted features in the trace have to be updated, the update rule of linear TD learning becomes

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t z_{t+1} \tag{21}$$

which is known as the TD(λ) algorithm. The equivalence of eligibility trace updates (sometimes referred to as *backward view*) and the weighted k-step lookahead reward (*forward view*) can be recognized by writing out the parameter updates for two succeeding timesteps. In addition, the formal proof for a more general case is available in Section 7.3 of [Maei, 2011]. The reader is referred to [Sutton and Barto, 1998] for further discussions and illustrations of eligibility traces. The concept of eligibility traces and multistep-lookahead has been used to develop several extensions of major TD Learning algorithms such as LSTD(λ) [Scherrer and Geist, 2011] or GTD(λ) [Maei, 2011] as the extension of TDC.

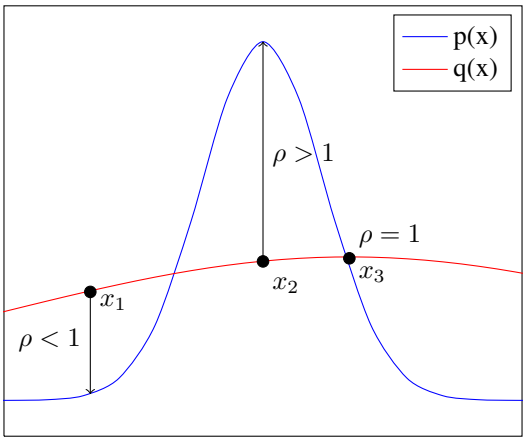

Figure 5: Importance Weighting: Samples drawn from $q(x)$ are reweighted by ρ to appear as drawn from $q(x)$

3.4 Off-Policy Extension by Importance Weighting

So far, the problem of estimating the value function V^π was addressed, given realizations $(s_t, a_t, r_t; t = 1 \dots T)$ of the MDP, where the actions are chosen according to the policy π. However, there exist many applications, where we want to know V^π, but only have samples with actions chosen by a different policy, for example when we search for the best policy for a robot with large operation costs. Then it is unfavorable to evaluate different policies on real hardware. Instead the quality of several policies has to be estimated given only data for a single one.

The described scenario is referred to as *off-policy value function prediction* and formally defined by: Given realizations $(s_t, a_t, r_t; t = 1 \dots T)$ of an MDP $\mathcal{M}$ and a policy π_B (*behavior policy*) , the goal is to find the value function of $\mathcal{M}$ for a different *target policy* π_G.

The extension of temporal difference methods to the off-policy case is based on the idea of *Importance Sampling* (cf. Section 12.2.2 of [Koller and Friedman, 2009]). It describes the relationship of expectations of an arbitrary function f, with random variables of two different distributions q and p as inputs:

$$\mathbb{E}_p[f(X)] = \mathbb{E}_q\left[\frac{p(X)}{q(X)} f(X)\right] \approx \frac{1}{M} \sum_{i=1}^{M} \frac{p(x_i)}{q(x_i)} f(x_i), \tag{22}$$

which can be verified easily by writing out the expectations. This statement allows us to compute expected values according to some distribution p,while drawing samples from a different distribution q by reweighting each sample $x_i, i = 1 \dots M$ by the ratio $\rho = \frac{p(x_i)}{q(x_i)}$. See Figure 5 for a visualization. Data points x_1 in regions, where q is larger than p occur more frequently in the sample from q than from p and are down-weighted. In the orthogonal case x_2, the weights are larger than one and give underrepresented samples more importance.

We look exemplary at the introduction of importance weights in classical linear TD learning (Equation (7)) to see the assumptions and limitations of off-policy extensions in general. Consider the expected parameter update $\mathbb{E}_{\pi_G}[\delta_t \phi(s_t)]$ according to the target policy π_G. We assume, that the states s_t the observed data are drawn i.i.d from an arbitrary distribution μ, which does not depend on the dynamics of the MDP or the policy. Note, that this is a very strong restriction, which usually does not hold in practice. When we observe data by following a trajectory, the states s_t for $t > \bar{t}$ depend on the action $a_{\bar{t}}$ chosen by π_B. Yet, only then the probability $p(s_t, a_t, s_{t+1})$ for a transition

	Fixpoint	Complexity	Eligibility Traces	Off-Policy Conv.	Idea
TD	(MSPBE)	$O(n)$	TD(λ)	no	bootstrapped SGD of (MSE)
GTD	(MSPBE)	$O(n)$	-	yes	SGD of (NEU)
GTD2	(MSPBE)	$O(n)$	-	yes	SGD of (MSPBE)
TDC	(MSPBE)	$O(n)$	GTD(λ)	yes	SGD of (MSPBE)
RG	(MSBE) / (MSTDE)	$O(n)$	-	yes	SGD of (MSBE)
LSTD	(MSPBE)	$O(n^2)$	LSTD(λ)	yes	iterative Least-Squares model estimation

Table 1: Algorithm Overview

from s_t to s_{t+1} with action a_t decomposes into $P(s_{t+1}|s_t, a_t)\pi_G(a_t|s_t)\mu(s_t)$ and we can write

$$\mathbb{E}_{\pi_G}[\delta_t\phi(s_t)] = \int p(s_t, a_t, s_{t+1})\delta_t\phi(s_t)\,\mathrm{d}(s_t, a_t, s_{t+1}) \tag{23}$$

$$= \int P(s_{t+1}|s_t, a_t)\pi_G(a_t|s_t)\mu(s_t)\;\; \delta_t\phi(s_t)\,\mathrm{d}(s_t, a_t, s_{t+1}) \tag{24}$$

$$= \int P(s_{t+1}|s_t, a_t)\pi_B(a_t|s_t)\mu(s_t)\;\; \frac{\pi_G(a_t|s_t)}{\pi_B(a_t|s_t)}\delta_t\phi(s_t)\,\mathrm{d}(s_t, a_t, s_{t+1}) \tag{25}$$

$$= \mathbb{E}_{\pi_B}[\rho_t\delta_t\phi(s_t)], \tag{26}$$

with the weight ρ_t of timestep t given as $\rho_t = \frac{\pi_G(a_t|s_t)}{\pi_B(a_t|s_t)}$. Here, the second important assumption becomes apparent. The weight is only well-defined for $\pi_B(a_t|s_t) > 0$. So, each action that might be chosen by π_G must have at least some probability to be observed.

Equation (26) motivates that the gradient not only has to be scaled by the stepsize α_t but by $\alpha_t\rho_t$ for off-policy evaluation and the classical linear TD update becomes

$$\theta_{t+1} = \theta_t + \alpha_t\rho_t\delta_t\phi(s_t) \tag{27}$$

Note, that the on-policy case can be treated as a special case with $\pi_G = \pi_B$ and $\rho_t = 1, \forall t = 1 \dots T$. Extensions of all common TD learning algorithms work analogously. The update rules shown in Figure 3 already contain the off-policy weights. In Section 4.2 the off-policy behavior of the algorithms is discussed from a theoretical and practical point of view.

4 Comparison of Algorithms

4.1 Alternative: LSTD(λ)

Stochastic gradient descent algorithms are very efficient methods in terms of cpu runtime. Nevertheless, alternative TD approaches exist, that implement other optimization techniques and therefore have different properties. The most popular representative is *Least Squares Temporal Difference Learning (LSTD(λ)* [Boyan, 2002]. In the following, LSTD is introduced briefly by a rough sketch of its idea and is then compared to the stochastic gradient algorithms presented in this paper. This allows to estimate the advantages and drawbacks of single algorithms in context to each other.

As we have already seen when motivating (NEU) as an objective function, the TD solution satisfies

$$0 = \mathbb{E}[\delta_t\phi_t] = \mathbf{\Phi}^T\mathbf{D}(\mathbf{T}\mathbf{V}_\theta - \mathbf{V}_\theta) = \mathbf{\Phi}^T\mathbf{D}(\mathbf{R} + \gamma\mathbf{P}\mathbf{V}_\theta - \mathbf{V}_\theta) \tag{28}$$

$$= \mathbf{\Phi}^T\mathbf{D}\mathbf{R} + \mathbf{\Phi}^T\mathbf{D}(\gamma\mathbf{P} - \mathbf{I})\mathbf{\Phi}\theta. \tag{29}$$

So, the TD fixpoint can be understood as a solution of a system of linear equations $\mathbf{A}\theta = b$ with $A = \mathbf{\Phi}^T\mathbf{D}(\gamma\mathbf{P} - \mathbf{I})\mathbf{\Phi}$ and $b = -\mathbf{\Phi}^T\mathbf{D}\mathbf{R}$. As opposed to gradient methods, the least squares approach

builds explicit models of $\mathbf{A}$ and b and then determines θ by solving the linear equation system robustly with Singular Value Decomposition [Klema and Laub, 1980]. This corresponds to building an empirical model of the MDP in memory. As we will see in the following sections, this model-based approach has both advantages and drawbacks compared to model-free gradient methods. An efficient recursive version of LSTD(λ) is proposed in [Yu, 2010], which directly updates $\mathbf{M} = A^{-1}$ and therefore avoids solving the linear equation system. Even though it requires setting an initial parameter ϵ for $\mathbf{M}_0 = \epsilon\mathbf{I}$, choosing any large value for ϵ worked well in our experiments. Since this paper focuses on fast algorithms, we only use the recursive version of LSTD(λ) for comparison. Its update rule extended by importance weighting and eligibility traces is shown in Algorithm 6.

4.2 Convergence Guarantees

Most important for the application of an algorithm is that it works reliably, at least when it is run long enough. Therefore, we first compare temporal difference methods by their convergence behavior. Apart from LSTD(λ), the algorithms are based on the idea of stochastic gradient descent and use step sizes to follow the gradient. As such, they are instances of Stochastic Approximation [Robbins and Monro, 1951] and appropriate step size schedules $(\alpha_t, t \leq 0)$ are supposed to satisfy

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty. \tag{30}$$

The foundations of stochastic approximation are used to prove convergence in the on-policy setting for each discussed algorithm to a well defined fixed point assuming valid step size schedules (see Table 1 for an overview). In [Tsitsiklis and Roy, 1997] is is shown that Linear TD(λ) converges even for linearly dependent feature representations. The proof of convergence of the residual gradient method for two independent samples of the successor state to the (MSBE) fixpoint can be found in [Baird, 1995], while [Maei, 2011] identifies (MSTDE) as fixed point for a single successor state samples. The recently proposed gradient temporal difference methods GTD, GTD2 and TDC, need two step size schedules α_t and β_t for which the conditions (30) have to hold. For GTD and GTD2 it is further required that $\alpha_t, \beta_t \in (0, 1]$ and that there exists a $\mu > 0$ such that $\beta_t = \mu\alpha_t$. Then convergence to the (MSPBE) fixpoint can be shown [Sutton et al., 2009a], [Sutton et al., 2009b]. The same results hold for TDC, except that $\lim_{t\to\infty} \frac{\alpha_t}{\beta_t} = 0$ is required.

The major motivation of the family of new gradient TD methods was the potential divergence of linear TD for off-policy prediction. In fact, the new algorithms are proven to converge in the off-policy case under the i.i.d. and exploration assumptions discussed in Section 3.4. In addition, the residual gradient method [Baird, 1995] and LSTD(λ) [Yu, 2010] are shown to converge in this setting. A common example for off-policy behavior of TD approaches is the *Baird-Star* shown in Figure 6. Experimental results in [Sutton et al., 2009b] reveal divergence of linear TD(0) and convergence of TDC using constant step sizes and "synchronous sweeps".

In contrast, we were not assuming knowledge of $\mathbb{E}[\delta\phi(s_t)]$ and used standard stochastic updates for our own investigations. The results in Figure 7 show, that the findings of [Sutton et al., 2009b] could be reversed. With appropriate choices of constant step sizes both TDC and linear TD(0) diverge resp. converge reliably in all 200 independent runs. Even for step size schedules satisfying the convergence conditions in (30), arbitrary behavior can be produced for TDC (blue and cyan curves). What seems surprising at first has two reasons: first, limited accuracy of floating point arithmetics, and second, the violation of i.i.d. off-policy assumption. In fact, the data was generated following a trajectory as it is the scenario relevant in practice. The results raise the question if the new gradient approaches have indeed better off-policy convergence properties in practice or if the proof assumptions are too strong.

Even if all algorithms found their fixed point, that would not answer the question which algorithm produces the best solution as none is optimizing the (MSE) directly. It is of active research interest which fixed points are superior in practice. [Sutton et al., 2009b] argues that the TD solution, i.e. the (MSPBE) fixed point should be preferred over the (MSBE) one. Yet, [Scherrer, 2010] suggests that TD solution is slightly better most often but, fails in some cases drastically and so the residual gradient fixed point should be preferred on average.

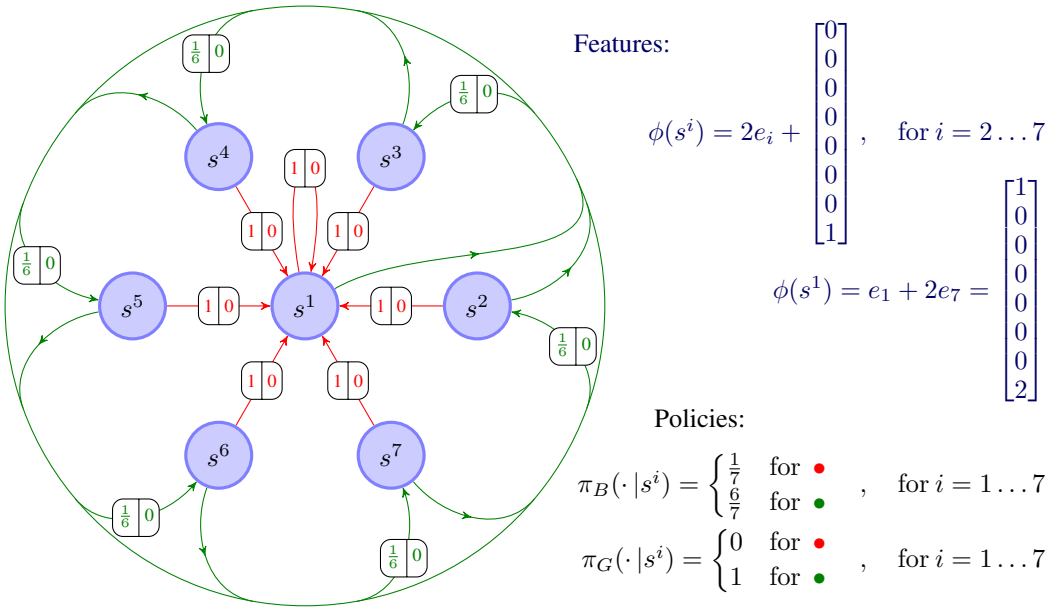

Figure 6: 7-State Baird Star MDP: off-policy example problem from [Baird, 1995]. Ergodic Markov decision process with a green and red action and a discount factor of $\gamma = 0.99$. The behavior policy choses with probability $\frac{1}{7}$ the red action which always transitions into the center state. If the green action was chosen, any vertices may be the state. The value function has to be computed for the target policy which picks the green action deterministically. The states are represented by an identity vector that is augmented by an additional dimension (1 for vertices, 2 for the center state).

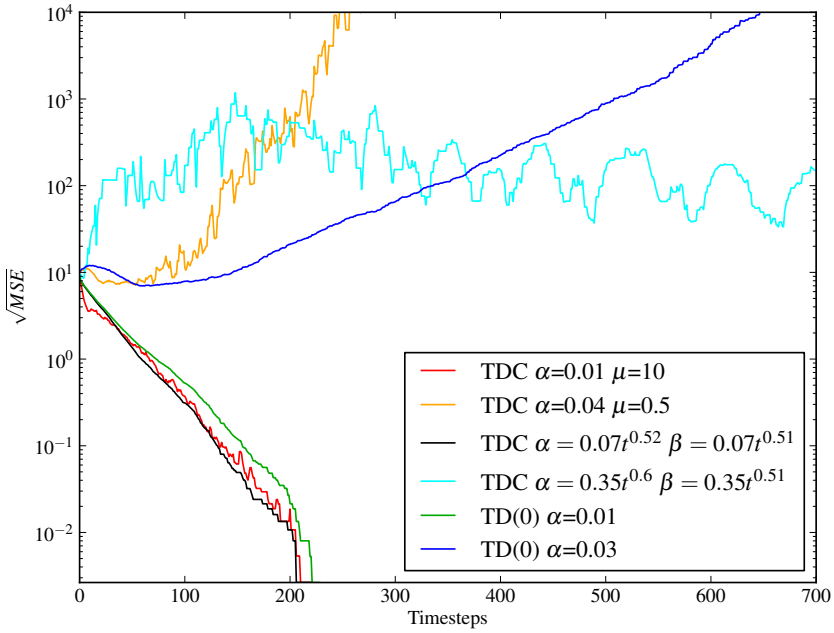

Figure 7: Convergence Comparison for Baird Star MDP (Fig 6). The graphs shown the root of the (MSE) over the number of observed transitions. The results are the mean of 200 independent runs. The standard errors are omitted for clarity due to the divergent behavior of some graphs.

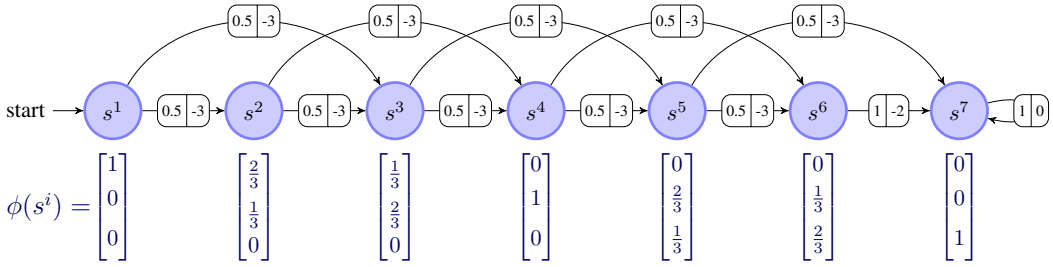

Figure 8: 7-State Boyan Chain MDP from [Boyan, 2002]

4.3 Convergence Speed

Besides stability and final output, it is of particular interest for reinforcement learning algorithms of how fast the desired output is reached. As TD methods are often applied in online learning scenarios without predefined start and end points, intermediate results are important. We base the discussion of convergence speed on our own experiments as well as results from related literature.

Two examples of MDPs are commonly used for empirical convergence speed analysis (cf. [Sutton and Barto, 1998, Boyan, 2002, Maei, 2011]). Those are the already introduced Random Walk Chain in Figure 4 and the *Boyan Chain*, that always contains only a single action but may have a varying number of states. A version with seven states is visualized in Figure 8. An episode always starts in s^1 and terminates when the other end of the chain is reached. At each timestep the system transitions with equal probability either one or two states further right with reward -3. Only in the second to last state, the system deterministically goes to the terminal state with reward -2. Just like in the random walk example, the true value function increases linearly from $V^\pi(s^1) = -24$ to $V^\pi(s^7) = 0$ for $\gamma = 1$. The states are represented by local linear features ϕ as shown in Figure 8.

[Sutton et al., 2009b] compares the convergence performance of recent gradient methods – TDC, GTD, GTD2 – to the classical linear TD(0) approach. We reproduced the experiment setting and included RG and LSTD(0) in the comparison. The results for the 7 state random walk MDP are shown in Figure 11 and Figure 10 visualized convergence performance on a Boyan Chain with 14 states and a 4-dimensional feature representation.

Each method is shown with optimal parameters found by an elaborate search. As in [Sutton et al., 2009b] the search was restricted to constant step sizes, but we used a finer grained parameter space. We observed throughout all gradient methods that the choice of constant step-sizes is always a tradeoff between fast initial descent and convergence accuracy. Exemplarily the detailed results for TD(0) for varying α are visualized in Figure 9. While small step sizes produce very accurate results in the long run, they converge slowly. Large step sizes decrease the root of (MSPBE) very fast but then stay at a high level with large variance. This observation motivated to chose the optimal step sizes as follows: Optimal step sizes let the method converge to a solution with $\sqrt{\text{MSPBE}(\theta)} \leq 0.5$ for the Boyan chain, resp. $\sqrt{\text{MSPBE}(\theta)} \leq 0.04$ for the random walk, and have the minimum accumulated $\sqrt{\text{MSPBE}(\theta)}$ during the first 100 episodes.

Our results confirm the findings of [Sutton et al., 2009b] that GTD has the slowest convergence rate of all methods. In addition, we found the performance of GTD2 and TDC to be similar throughout all experiments. TDC is slightly faster for the random walk, but that depends heavily on the exact step size schedules α and $\beta = \alpha\mu$. While both methods could outperform TD(0) significantly for the Boyan chain, the classical approach is en par with TDC considering the random walk example.

LSTD(0) converges in all experiments significantly faster and with extremely high accuracy compared to the gradient approaches. This results validate the high data-efficiency of LSTD as a model-based approach [Boyan, 2002]. The performance of Residual Gradient depends strongly on the chosen MDP. While showing slow convergence for the Boyan chain, it finds the perfect parameters for the random walk with tabular features already after 50 episodes. This is most surprising since it does not minimize the (MSPBE) directly (but still converges to the fixed point as $V^\pi \in \mathcal{H}_\phi$ for both examples).

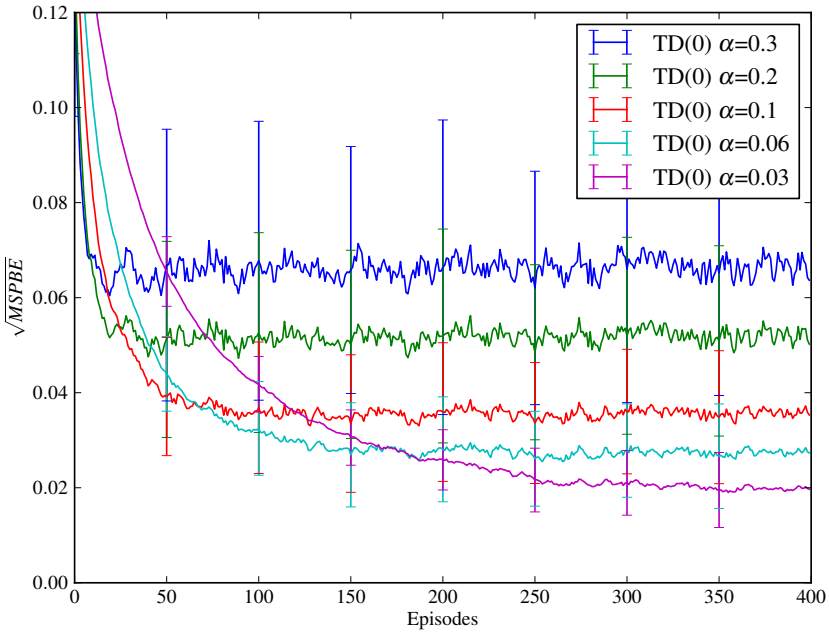

Figure 9: Dependency of Step Size on Convergence Speed of TD(0) for 7-State Random Walk MDP (Figure 4) with tabular features. The results are averages over 200 independent trials.

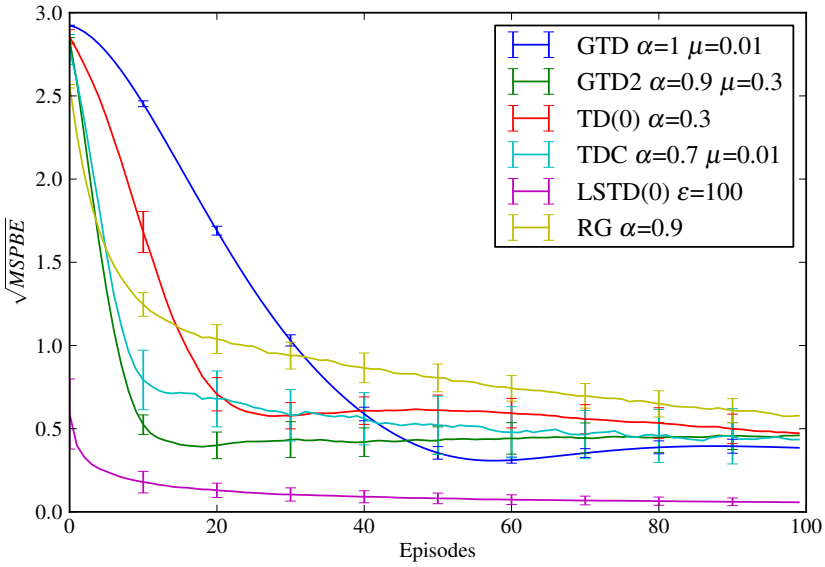

Figure 10: Convergence Speed Comparison for a Boyan Chain MDP (Figure 8) with 14 states and a 4-dimensional state representation. The results are averages over 200 independent trials.

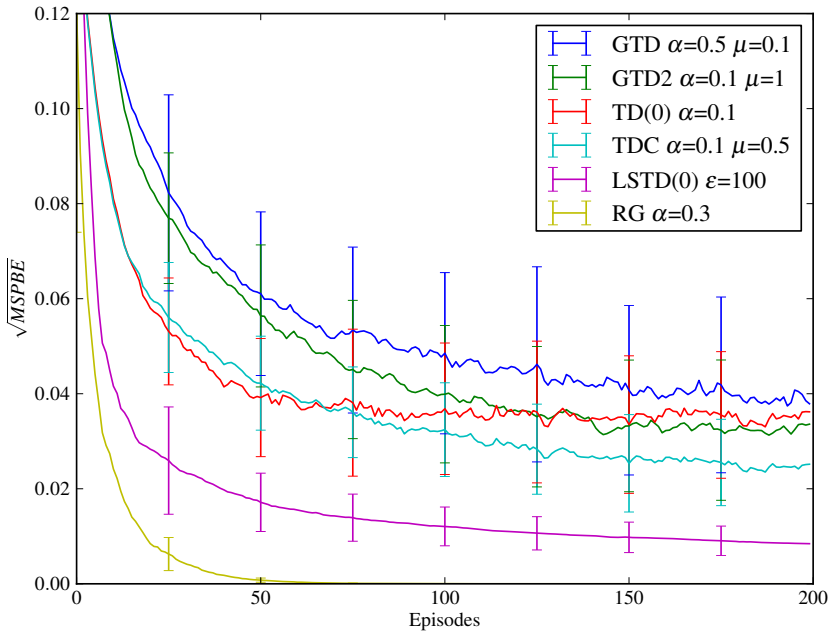

Figure 11: Convergence Speed Comparison for 7-State Random Walk MDP (Figure 4) with tabular features. The results are averages over 200 independent trials.

The results could be mistaken that LSTD(λ) is always significantly faster than gradient approaches. It is important to notice, that the graphs do not show actual computing time. As LSTD(λ) has to store and update an $m \times m$ matrix each timestep, it has a time and space complexity of $O(m^2)$, where m is the dimension of the feature representation. In contrast, all other methods work only on vectors of length m and therefore have complexities of $O(m)$. While this does not play a role for the example problems discussed to far, we observed the computing advantages of gradient approaches in experiments on randomly generated MDPs with larger numbers of states and features. We trained all methods for 100 timesteps on MDPs with $m = |\mathcal{S}| = 100, 500, 100, 2500, 5000$. While the computing time of gradient TD learners is always negligible, LSTD takes for 500 states already 0.1s increasing by 0.4s and 4s up to 13s for 5000 states. The actual times depend heavily on the implementation, but still indicate that LSTD is far too slow for real world online system with even a small number of features.

5 Conclusion & Further Reading

This paper had the focus of familiarizing novices of temporal difference learning with the recently developed fast gradient descent algorithms GTD(2) and TDC. Instead of going into specific details, the main concepts that lead to these methods were identified and explained such that readers with sufficient background in math and machine learning can understand what differentiates the approaches. The presentation of the main components, stochastic gradient descent, eligibility traces and importance weighting for off-policy prediction, was followed by a comparison of the algorithms in terms of convergence guarantees and speed. Results of own experiments indicated the following main conclusions:

- For offline prediction with enough computing time LSTD(λ) should be preferred to gradient methods due to data-efficiency.
- Online learning scenarios require gradient methods since their time complexity is linear in the number of states.
- GTD2 and TDC should be preferred to GTD.

- TDC outperforms classical TD, but may require a more elaborate parameter search due to the additional step size β.
- The relevance of off-policy convergence guarantees of TDC and GTD(2) is questionable, since the assumption of i.i.d. samples usually does not hold in practice so that TDC actually diverges.

As all introductory paper, a trade-off between elaborateness and space needed to be found. Many concepts could only be sketched briefly. So, we encourage the reader to have a look at our reimplementation of each algorithm in Python for a better understanding of the topics. As far as we know it is the first publicly available implementation of TDC and GTD. The source can be found at https://github.com/chrodan/tdlearn.

This paper has prepared the reader to be able to understand further material in the area of temporal difference learning. That can be the original papers of the discussed algorithms [Sutton and Barto, 1998, Sutton et al., 2009b, Sutton et al., 2009a, Baird, 1995, Boyan, 2002] or other more elaborate introductions to algorithms of Reinforcement Learning in general such as [Szepesvári, 2010]. In addition, [Geist and Pietquin, 2011] may be of interest. It has similar goal as this paper – give an overview over TD methods for value prediction – but addresses readers with stronger background in TD learning, since it introduces a unified view. Finally, we refer the reader to the PhD thesis of Hamid Reza Maei [Maei, 2011], where TDC and GTD are extended to work with non-linear function approximations. In addition, the Greedy-GQ(λ) method, an off-policy control approach building directly on the ideas of GTD and TDC makes the thesis particular interesting.

References

[Baird, 1995] Baird, L. (1995). Residual Algorithms : Reinforcement Learning with Function Approximation. *International Conference on Machine Learning*.

[Balakrishna et al., 2010] Balakrishna, P., Ganesan, R., and Sherry, L. (2010). Accuracy of reinforcement learning algorithms for predicting aircraft taxi-out times: A case-study of Tampa Bay departures. *Transportation Research Part C: Emerging Technologies*, 18(6):950–962.

[Boyan, 2002] Boyan, J. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246.

[Frank et al., 2008] Frank, J., Mannor, S., and Precup, D. (2008). Reinforcement learning in the presence of rare events. In *Proceedings of the 25th international conference on Machine learning*, number January, pages 336–343. McGill University, Montreal, ACM.

[Geist and Pietquin, 2011] Geist, M. and Pietquin, O. (2011). Parametric Value Function Approximation: a Unified View. *Adaptive Dynamic Programming and Reinforcement Learning*, pages 9–16.

[Klema and Laub, 1980] Klema, V. and Laub, A. (1980). The Singular Value Decomposition: Its Computation and Some Applications. *IEEE Transactions On Automatic Control*, 25(2).

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. Adaptive Computation and Machine Learning. MIT Press.

[Lucian Busoniu, Robert Babuska, Bart De Schutter, 2010] Lucian Busoniu, Robert Babuska, Bart De Schutter, D. E. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press.

[Maei, 2011] Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms*. PhD thesis, University of Alberta.

[Riedmiller and Gabel, 2007] Riedmiller, M. and Gabel, T. (2007). On Experiences in a Complex and Competitive Gaming Domain: Reinforcement Learning Meets RoboCup. *2007 IEEE Symposium on Computational Intelligence and Games*, (Cig):17–23.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407.

[Rosenblatt, 1971] Rosenblatt, M. (1971). *Markov Processes. Structure and Asymptotic Behavior*. Springer-Verlag.

[Russell and Norvig, 2003] Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, volume 60 of *Prentice Hall Series In Artificial Intelligence*. Prentice Hall.

[Scherrer, 2010] Scherrer, B. (2010). Should one compute the Temporal Difference fix point or minimize the Bellman Residual? The unified oblique projection view. *International Conference on Machine Learning*.

[Scherrer and Geist, 2011] Scherrer, B. and Geist, M. (2011). Recursive Least-Squares Learning with Eligibility Traces. In *European Workshop on Reinforcement Learning*.

[Silver and Sutton, 2007] Silver, D. and Sutton, R. (2007). Reinforcement Learning of Local Shape in the Game of Go. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1053–1058.

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

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*, volume 9 of *Adaptive computation and machine learning*. MIT Press.

[Sutton et al., 2009b] Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvari, C., and Wiewiora, E. (2009b). Fast gradient-descent methods for temporal-difference learning with linear function approximation. *Proceedings of the 26th Annual International Conference on Machine Learning ICML 09*.

[Szepesvári, 2010] Szepesvári, C. (2010). Algorithms for Reinforcement Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103.

[Tsitsiklis and Roy, 1997] Tsitsiklis, J. N. and Roy, B. V. (1997). An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions On Automatic Control*, 42(5):674–690.

[Yu, 2010] Yu, H. (2010). Convergence of least squares temporal difference methods under general conditions. *Proc. of the 27th ICML, Haifa, Israel*.