Journal of Machine Learning Research 15 (2014) 809-883 Submitted 5/13; Revised 11/13; Published 3/14

Policy Evaluation with Temporal Differences: A Survey and Comparison

Christoph Dann CDANN@CDANN.DE
Gerhard Neumann GERI@ROBOT-LEARNING.DE
Technische Universität Darmstadt
Karolinenplatz 5
64289 Darmstadt, Germany

Jan Peters* MAIL@JAN-PETERS.NET
Max Planck Institute for Intelligent Systems
Spemannstraße 38
72076 Tübingen, Germany

Editor: Peter Dayan

Abstract

Policy evaluation is an essential step in most reinforcement learning approaches. It yields a value function, the quality assessment of states for a given policy, which can be used in a policy improvement step. Since the late 1980s, this research area has been dominated by temporal-difference (TD) methods due to their data-efficiency. However, core issues such as stability guarantees in the off-policy scenario, improved sample efficiency and probabilistic treatment of the uncertainty in the estimates have only been tackled recently, which has led to a large number of new approaches.

This paper aims at making these new developments accessible in a concise overview, with foci on underlying cost functions, the off-policy scenario as well as on regularization in high dimensional feature spaces. By presenting the first extensive, systematic comparative evaluations comparing TD, LSTD, LSPE, FPKF, the residual-gradient algorithm, Bellman residual minimization, GTD, GTD2 and TDC, we shed light on the strengths and weaknesses of the methods. Moreover, we present alternative versions of LSTD and LSPE with drastically improved off-policy performance.

Keywords: temporal differences, policy evaluation, value function estimation, reinforcement learning

1. Introduction

Policy evaluation estimates a value function that predicts the accumulated rewards an agent following a fixed policy will receive after being in a particular state. A policy prescribes the agent's action in each state. As value functions point to future success, they are important in many applications. For example, they can provide failure probabilities in large telecommunication networks (Frank et al., 2008), taxi-out times at big airports (Balakrishna et al., 2010) or the importance of different board configurations in the game Go (Silver et al., 2007). Such value functions are particularly crucial in many *reinforcement*

∗. Also at *Technische Universität Darmstadt, Karolinenplatz 5, Darmstadt, Germany.*

learning methods for learning control policies as one of the two building blocks constituting *policy iteration*. In policy iteration, an optimal policy is obtained by iterating between the value prediction for states (and sometimes actions) given the agent's current policy, that is, *policy evaluation*, and improving the policy such that it maximizes the value of all states predicted by the current value function, that is, *policy improvement*. Policy-iteration-based reinforcement learning has yielded impressive applications in robot soccer (Riedmiller and Gabel, 2007), elevator control (Crites and Barto, 1998) and game-playing such as Checkers (Samuel, 1959), Backgammon (Tesauro, 1994) and Go (Gelly and Silver, 2008). For sufficiently accurate value function estimates, policy iteration frequently converges to the optimal policy. Hence, a reliable and precise estimator of the value function for a given policy is essential in reinforcement learning and helpful in many applications.

However, obtaining accurate value function estimates is not a straightforward supervised learning problem. Creating sufficient data for obtaining the value function by regression would require a large number of roll-outs (state-action-reward sequences) in order to acquire the accumulated reward for each considered state. As the variance of the accumulated reward frequently grows with the time horizon, exhaustive number of data points would be required. To circumvent this issue, the idea of bootstrapping has been proposed, that is, the current estimate of the value function is used to generate the target values for learning a new estimate of the value function. In expectation, the sum of the current reward and the discounted value of the next state should match the value of the current state. Hence, their difference becomes an error signal for the value function estimator. The resulting approaches are called temporal-difference methods from their introduction in Sutton (1988). Temporal-difference methods have received a tremendous attention in the last three decades and had a number of important successes including the ones mentioned in the previous paragraph.

While temporal-difference methods have been successful, they have not been understood well for a long time (Tsitsiklis and van Roy, 1997; Schoknecht, 2002), they were data-inefficient (Bradtke and Barto, 1996) , and were not stable if used with function approximation in the off-policy case (Baird, 1995). In the off-policy scenario, the value function is estimated from a data set that was generated from another policy than the one we want to evaluate, which is crucial for data re-use in policy iteration. Recently, there has been a large number of substantial advances both in our understanding of temporal-difference methods as well as in the design of novel estimators that can deal with the problems above. These methods are currently scattered over the literature and usually only compared to the most similar methods. In this survey paper, we attempt at presenting the state of the art combined with a more comprehensive comparison.

This survey has two major contributions. First, we are going to present a principled and structured overview on the classic as well as the recent temporal-difference methods derived from general insights. Second, we are comparing these methods in several meaningful scenarios. This comprehensive experimental study reveals the strengths and weaknesses of temporal-difference methods in different problem settings. These insights on the behavior of current methods can be used to design improvements which overcome previous limitations as exemplified by the alternative off-policy reweighting strategy for LSTD and LSPE proposed in this paper. The remainder of this paper is structured as follows: Sections 1.1 and 1.2 introduce the required background for this paper on Markov decision processes and value functions. As the paper aims at complementing the literature, especially the book

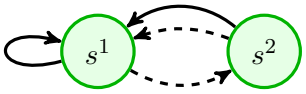


Figure 1: Stationary Distributions for Different Policies. The MDP has deterministic transitions depending on the state (1 or 2) and the action (solid or dashed) illustrated by the arrows. Taking for example the dashed action in state 1 moves the agent always to state 2. A policy which always chooses the solid action leaves the agent always in state 1, that is, $d_{\text{solid}} = [1, 0]^T$, while the dashed counterpart makes the agent alternate between the two states ($d_{\text{dashed}} = [\frac{1}{2}, \frac{1}{2}]^T$). If the agent takes the solid action with probability α, the steady state distribution is given by $d_\alpha = [\frac{1}{2} + \frac{1}{2}\alpha, \frac{1}{2} - \frac{1}{2}\alpha]^T$.

by Sutton and Barto (1998), we illustrate the concept of temporal-difference methods for policy evaluation already in Section 1.2. In Section 2, we present a structured overview of current policy evaluation methods based on temporal differences. This overview starts out by presenting the core objective functions that underlie the various different temporal-difference-based value function methods. We show how different algorithms can be designed by using different optimization techniques, such as stochastic gradient descent, least-squares methods or even Bayesian formulation, resulting in a large variety of algorithms. Furthermore, we illustrate how these objectives can be augmented by regularization to cope with overabundant features. We present important extensions of temporal-difference learning including eligibility traces and importance reweighting for more data-efficiency and estimation from off-policy samples. As Section 2 characterizes methods in terms of design decisions, it also sheds light on new combinations not yet contributed to the literature. In Section 3, we first present a series of benchmark tasks that are being used for comparative evaluations. We focus particularly on the robustness of the different methods in different scenarios (e.g., on-policy vs. off-policy, continuous vs. discrete states, number of features) and for different parameter settings (i.e., the open parameters of the algorithms such as learning rates, eligibility traces, etc). Subsequently, a series of important insights gained from the experimental evaluation is presented including experimental validation of known results as well as new ones which are important for applying value-function estimation techniques in practice. The paper is concluded in Section 4 with a short summary and an outlook on the potential future developments in value-function estimation with temporal differences.

1.1 Notation and Background on Markov Decision Processes

The learning agent's task is modeled as a *Markov decision process* (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R)$. At each discrete time step $t = 0, 1, 2 \ldots$, the system is in a state $s_t \in S$ and the agent chooses an action $a_t \in \mathcal{A}$. The state of the next time step is then determined by the transition model $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, that is, $\mathcal{P}(s_{t+1}|a_t, s_t)$ is the conditional probability (density) for transitioning from s_t to s_{t+1} with action a_t. After each transition, the agent receives a reward $r_t = R(s_t, a_t)$ specified by the deterministic *reward function* $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We distinguish between discrete systems and continuous systems. While continuous systems

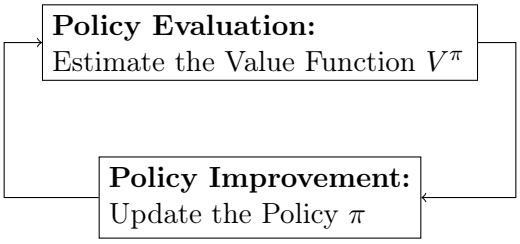


Figure 2: Policy Iteration Algorithm

have infinitely many states (and actions), discrete systems are usually restricted to a finite number of states. For notational simplicity, we mostly treat $\mathcal{S}$ and $\mathcal{A}$ to be finite sets in the remainder of this paper. Nevertheless, the analysis presented in this paper often generalizes to continuous/infinite state-spaces.

The behavior of the learning agent within the environment, that is, the action-selection strategy given the current state, is denoted by a *policy* π. A stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines a probability distribution over actions given a state s_t. The agent samples from π to select its actions. Stochasticity of the policy promotes state exploration, a key component of robust policy learning. However, in certain cases a deterministic policy can be easier to handle. Such a policy can be treated as a deterministic function $\pi : \mathcal{S} \to \mathcal{A}$ with $a_t = \pi(s_t)$.

While we also consider episodic Markov decision processes in examples and experiments, we concentrate on ergodic MDPs for the formal part to keep the theoretical analysis concise. Ergodic Markov decision processes do not terminate and the agent can interact with its environment for an infinite time. Their underlying stochastic processes have to be ergodic, which, in highly simplified terms, means that every state can be reached from all others within a finite amount of time steps (for details and exact definitions see for example the work of Rosenblatt, 1971). If these assumptions hold, there exists a *stationary distribution* d^π over $\mathcal{S}$ with $d^\pi(s') = \sum_{s,a} \mathcal{P}(s'|s,a)\pi(a|s)d^\pi(s)$. This distribution yields the probability of the process being in state s when following policy π, that is, sampled states from the MDP with policy π are identically distributed samples from d^π. While they are not (necessarily) independently drawn, ergodicity ensures that the strong law of large numbers still holds. Formally, MDPs do not need to have unique limiting distributions. Instead, a distribution defined as $d^\pi(s) = \mathbb{E}_{\pi,\mathcal{P}}\left[\sum_{t=0}^{\infty} 1_{\{s_t=s\}}\right]$ would suffice in most cases. For fixed policies π, we can rewrite the definition d^π more concisely as $d^\pi(s) = \mathbb{E}_{\mathcal{P}^\pi}\left[d^\pi(s)\right]$, where $\mathcal{P}^\pi$ denotes the state transition distribution

$$\mathcal{P}^\pi(s_{t+1}|s_t) = \sum_{a_t} \mathcal{P}(s_{t+1}|a_t, s_t)\pi(a_t|s_t).$$

Marginalizing out the action reduces the MDP to a Markov chain. Even though the actions are marginalized out, the policy affects $\mathcal{P}^\pi$ and, thus, the stationary distribution is highly dependent on π. See Figure 1 for an example.

Reinforcement learning aims at finding a policy that maximizes the *expected (total discounted) future reward*

$$J(\pi) = \mathbb{E}_{\mathcal{P},\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right].$$

The discount factor $\gamma \in [0, 1)$ controls the considered timespan or planning horizon. Small discount factors emphasize earlier rewards while rewards in the future are becoming less relevant with time.

A common family of iterative reinforcement learning algorithms for finding the optimal policy is policy iteration. Policy iteration algorithms alternate between a *policy evaluation* and a *policy improvement* step (see Figure 2). In the policy evaluation step, the value function $V^\pi : \mathcal{S} \to \mathbb{R}$ for the current policy is estimated. The value function corresponds to the expected accumulated future reward

$$V^\pi(s) = \mathbb{E}_{\mathcal{P},\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s\right], \tag{1}$$

given that the process started in state s, and that the actions are chosen according to policy π. Hence, the value function evaluates the policy in each state. It allows the subsequent policy improvement step to obtain a policy which chooses actions that move the agent most likely in states with the highest values.

1.2 Problem Statement: Efficient Value Function Estimation

This paper discusses different approaches to estimate the value function in Equation (1) while observing the agent interacting with its environment. More formally, the problem of *value-function estimation* can be defined as follows:

The value function of a target policy π_G and a given MDP $\mathcal{M}$ is estimated based on the data set $\mathcal{D} = \{(s_t, a_t, r_t; t = 1 \dots t_f), (s_t, a_t, r_t; t = 1 \dots t_f), \dots\}$ sampled from the MDP $\mathcal{M}$ and a behavior policy π_B. The data set $\mathcal{D}$ may consist of one or more roll-outs $(s_t, a_t, r_t; t = 1 \dots t_f)$. We distinguish between *on-policy* estimation ($\pi_B = \pi_G$) and *off-policy* estimation ($\pi_B \neq \pi_G$). The latter scenario is particularly appealing for policy iteration, as we can re-use samples from previous policy evaluation iterations for the current value function.

To illustrate major challenges of value-function estimation we consider a classic 2D grid-world example shown in Figure 3, a simple benchmark task often used in reinforcement learning. To estimate the value of the agent's position, we have to compute the expectation in Equation (1). We can approximate this value directly with Monte-Carlo methods, that is, take the average of the accumulated reward computed for several roll-outs starting from this position (Sutton and Barto, 1998, Chapter 5). However, the variance of the accumulated reward will be huge as the stochasticity of each time-step often adds up in the accumulated rewards. For example one roll-out may yield a high reward as the agent always moves in the directions prescribed by the policy, while another roll-out may yield very low reward as the agent basically performs a random walk due to the transition probabilities of the MDP. Hence, even if we have a model of the MDP to simulate the agent until future rewards are sufficiently discounted, the value estimate of Monte-Carlo methods is typically highly inaccurate in any reasonable limit of roll-outs.

The crux is the dependency of the state value on future rewards, and subsequently on the state after many time-steps. The problem simplifies with decreasing discount factor γ and reduces to standard supervised learning for $\gamma = 0$ (estimate immediate reward $\mathbb{E}[r_t | s_t = s]$). Bootstrapping is an approach to circumvent the problems of long-time dependencies using

a recursive formulation of the value function. This recursion can be obtained by comparing Equation (1) for two successive time-steps t and $t+1$

$$V^\pi(s) = \mathbb{E}_{\mathcal{P},\pi}\Big[r(s_t, a_t) + \gamma V^\pi(s_{t+1})\Big| s_t = s\Big]. \tag{2}$$

This so-called *Bellman equation*[1] holds true for arbitrary MDPs $\mathcal{M}$, discount factors γ and policies π. This basic insight allows us to update the value estimate of the current state based on the observed reward r_t and the value estimate for the successor state s_{t+1}. In the long run, the estimate is changed such that the difference of the values of temporally subsequent states (temporal difference) matches the observed rewards in expectation. This bootstrapping approach is the foundation for efficient value-function estimation with temporal-difference methods, as it drastically reduces the variance of the estimator. Yet, it may also introduce a bias (cf. Section 2.1).

To simplify notation we will write V^π as a $m = |\mathcal{S}|$ dimensional vector $\boldsymbol{V}^\pi$ which contains $V^\pi(s^i)$ at position i for a fixed order $s^1, s^2, \dots s^m$ of the states. Using the same notation for the rewards, that is, $\boldsymbol{R}^\pi \in \mathbb{R}^m$ with $\boldsymbol{R}^\pi_i = \mathbb{E}_\pi\big[r(s^i, a)\big]$, the Bellman equation can be rewritten as

$$\boldsymbol{V}^{\boldsymbol{\pi}} = \boldsymbol{R}^{\boldsymbol{\pi}} + \gamma \boldsymbol{P}^{\boldsymbol{\pi}} \boldsymbol{V}^{\boldsymbol{\pi}} =: T^\pi \boldsymbol{V}^{\boldsymbol{\pi}}. \tag{3}$$

Here, the transition matrix $\boldsymbol{P}^{\boldsymbol{\pi}} \in \mathbb{R}^{m\times m}$ of policy π contains the state transitions probabilities $P^\pi_{ij} = \sum_a \mathcal{P}(s^i|s_j, a)\pi(a|s_j)$. As we can see, the Bellman equation specifies a fixpoint of an affine transformation $T^\pi : \mathbb{R}^m \to \mathbb{R}^m$ of $\boldsymbol{V}^\pi$ (*Bellman operator*). We will omit the policy superscripts in unambiguous cases for notational simplicity.

The depicted world in Figure 3 consists of $15 \times 15 = 225$ states, that is, we have to estimate 225 values. Yet, such a small world can only be used for highly simplified tasks. More realistic settings (such as street navigation) require much finer and larger grids or have to allow arbitrary continuous positions $s \in \mathbb{R}^2$. This requirement illustrates another inherent problem of value estimation, the curse of dimensionality, that is, the number of states $|\mathcal{S}|$ increases exponentially with the number of state variables. For example, if there are several moving objects in our grid world, the number of states $|\mathcal{S}|$ explodes. In a 15×15 grid world with one agent and 9 moving objects, we have to estimate $(15 \times 15)^{10} \approx 3^{23}$ values. Thus, we almost always need to resort to approximation techniques for the value function. The simplest and most common approach is a linear parametrization with parameter vector $\boldsymbol{\theta} \in \mathbb{R}^n$, that is,

$$V(s) \approx V_\theta(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s),$$

where $\boldsymbol{\phi}(s)$ defines features of the state s.

The feature function $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^n$ reduces the number of parameters which we need to estimate from m to n with $n \ll m$ but comes at the price of less precision. Hence, the choice of a feature representation is always a trade-off between compactness and expressiveness, where the latter means that there exists a V_θ that is close to V for all states.

Estimation techniques for alternative parametrizations of V exist, such as non-linear function approximation (e.g.,, see non-linear versions of GTD and TDC, Maei, 2011, Chapter 6) or automatically built representations (cf. Section 2.3). However, defining non-linear

1. Bellman would not have claimed this equation but rather the principle of optimality (source: personal correspondence with Bellman's former collaborators).

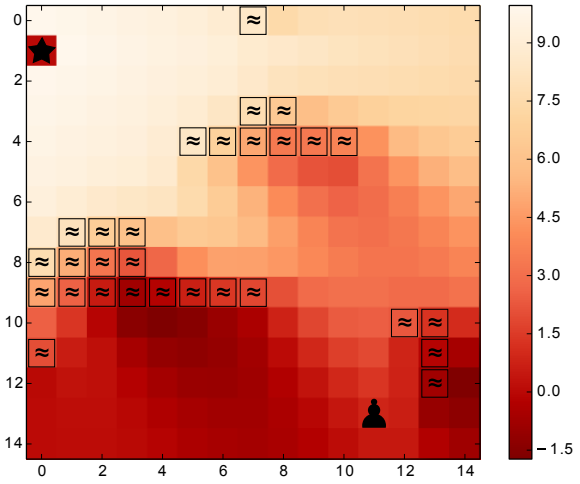


Figure 3: Classic 2D Grid-World Example: The agent ♟ obtains a positive reward (10) when it reaches the goal and negative (-2) ones when it goes through water ≈. The agent always chooses a direction (up, right, down left) that points towards the goal ★. With probability 0.8 the agent moves in that direction and with 0.2 in a random direction. We are interested in the value $V(s)$ of each possible position (state) of the agent with a discount of $\gamma = 0.99$. The value $V(s)$ is shown as an overlay.

function approximations by hand requires domain knowledge to an extent that is usually not available and the learning problem typically becomes non-convex, that is, the estimator may get stuck in local, but not global, optima. Therefore, this paper focuses on more commonly used linear function approximation.

After having identified temporal differences and function approximation as the key ingredients for efficient policy evaluation, we can concentrate on important properties of value function estimators. In robotics and many other fields, the data gathering process is very costly and time consuming. In such cases, algorithms need to be data efficient and yield accurate estimates already after few observations. In other applications, accurate models of the learning environment are available and observations can be generated efficiently by simulation. Hence, the focus is shifted to efficiency in terms of computation time. Computation time is also a limiting factor in online and real-time learning, which require to update the value estimations after each observation within a limited time frame.

2. Overview of Temporal-Difference Methods

Value estimation can be cast as an optimization problem, and, in fact, most temporal-difference methods are direct applications of optimization techniques. Hence, their characteristics are largely determined by (1) the chosen objective or cost function, and (2) how this function is optimized. We start by discussing different optimization objectives in Section 2.1 that build the basis of most theoretical results and define the quality of the value estimates

after enough observations. In Section 2.2, we introduce temporal-difference methods by grouping them according to the employed optimization technique as algorithms within a group share similar convergence speed and computational demands. To avoid cluttered notation and to put the focus on their intrinsic characteristics, we present the algorithms in their basic form and omit eligibility traces and importance weights for the off-policy case here (these are discussed in Section 2.4). Complete versions of the algorithms with available extensions can be found in Appendix C.

Reliable value estimates are the result of fruitful interplay between expressive feature descriptors $\boldsymbol{\phi}$ and suitable estimation algorithms. Yet, choosing appropriate feature functions poses a hard problem when insufficient domain knowledge is available. Several approaches have been proposed to simplify the generation and handling of features for temporal-difference methods. We review these recent efforts in Section 2.3. Finally, in Section 2.4, we discuss two important extensions applicable to most methods. The first extension are eligibility traces, which reduce bootstrapping and may increase convergence speed. Subsequently, we discuss importance reweighting for off-policy value-function estimation.

2.1 Objective Functions

We are interested in estimating parameters $\boldsymbol{\theta}$ that yield a value function $\boldsymbol{V}_\theta$ as close as possible to the true value function $\boldsymbol{V}^\pi$. This goal directly corresponds to minimizing the *mean squared error* (MSE)

$$\begin{aligned}\mathrm{MSE}(\boldsymbol{\theta}) &= \|\boldsymbol{V_\theta} - \boldsymbol{V^\pi}\|^2_{\boldsymbol{D}} = \sum_{i=1}^{m} d^\pi(s^i)[V_{\boldsymbol{\theta}}(s^i) - V^\pi(s^i)]^2 \\ &= [\boldsymbol{V_\theta} - \boldsymbol{V^\pi}]^T \boldsymbol{D}[\boldsymbol{V_\theta} - \boldsymbol{V^\pi}]. \end{aligned} \tag{4}$$

The weight matrix $\boldsymbol{D} = \mathrm{diag}[d^\pi(s^1), d^\pi(s^2), \ldots, d^\pi(s^m)]$ has the entries of the stationary distribution d^π on the diagonal and weights each error according to its probability. The true value function $\boldsymbol{V}^\pi$ can be obtained by Monte-Carlo estimates, that is, performing roll-outs with the current policy and collecting the long-term reward. However, the Monte-Carlo estimate of $\boldsymbol{V}^\pi$ requires a lot of samples as it suffers from a high variance.

The high variance can be reduced by eliminating the true value function $\boldsymbol{V}^\pi$ from the cost function. To do so, we can use *bootstrapping* (Sutton and Barto, 1998), where $\boldsymbol{V}^\pi$ is approximated by a one-step prediction based on the approximated value-function. Hence, we minimize the squared difference between the two sides of the Bellman equation (3) which corresponds to minimizing

$$\mathrm{MSBE}(\boldsymbol{\theta}) = \|\boldsymbol{V_\theta} - T\boldsymbol{V_\theta}\|^2_{\boldsymbol{D}}. \tag{5}$$

This objective, called the *mean squared Bellman error*, can be reformulated in terms of expectations using Equation (2)

$$\begin{aligned}\mathrm{MSBE}(\boldsymbol{\theta}) &= \sum_{i=1}^{n} d^\pi(s^i)[V_{\boldsymbol{\theta}}(s^i) - \mathbb{E}_{\mathcal{P},\pi}[r(s_t, a_t) + \gamma V_{\boldsymbol{\theta}}(s_{t+1})|\pi, s_t = s^i]]^2 \\ &= \mathbb{E}_d\big[\big(V_{\boldsymbol{\theta}}(s) - \mathbb{E}_{\mathcal{P},\pi}[r(s_t, a_t) + \gamma V_{\boldsymbol{\theta}}(s_{t+1})|\pi, s_t = s]\big)^2\big], \end{aligned} \tag{6}$$

where the outer expectation is taken with respect to the stationary distribution d^π and the inner one with respect to the state dynamics $\mathcal{P}$ of the MDP and the policy π. Let δ_t denote

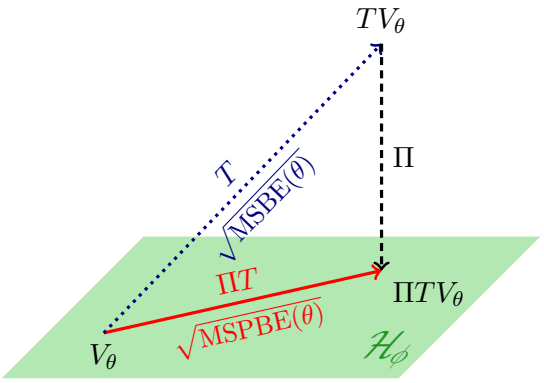


Figure 4: MSBE compares the current value estimate $\boldsymbol{V_\theta}$ to the T-transformed one $T\boldsymbol{V_\theta}$. In contrast, MSPBE is a distance in the space of parameterized functions $\mathcal{H}_{\boldsymbol{\phi}}$ and always smaller or equal than MSBE, as Π is an orthogonal projection. Figure adopted from Lagoudakis and Parr (2003).

the temporal-difference (TD) error which is given by the error in the Bellman equation for time step t

$$\delta_t = r(s_t, a_t) + \gamma V_{\boldsymbol{\theta}}(s_{t+1}) - V_{\boldsymbol{\theta}}(s_t) = r_t + (\gamma \boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t)^T \boldsymbol{\theta}, \tag{7}$$

with $\boldsymbol{\phi}_t = \boldsymbol{\phi}(s_t)$. Equation (6) can then be written concisely as $\text{MSBE}(\boldsymbol{\theta}) = \mathbb{E}_d[\mathbb{E}_{\mathcal{P},\pi}[\delta_t|s_t]^2]$. Using the second form of δ_t from Equation (7) to formulate the MSBE error as

$$\text{MSBE}(\boldsymbol{\theta}) = \mathbb{E}_d\left[\left(\left(\mathbb{E}_{\mathcal{P},\pi}[\gamma\boldsymbol{\phi}_{t+1}|s_t] - \boldsymbol{\phi}_t\right)^T \boldsymbol{\theta} + \mathbb{E}_{\mathcal{P},\pi}[r_t|s_t]\right)^2\right]$$

makes apparent that the MSBE objective corresponds to a linear least-squares regression model with inputs $\gamma\mathbb{E}_{\mathcal{P},\pi}[\boldsymbol{\phi}_{t+1}|s_t] - \boldsymbol{\phi}_t$ and outputs $-\mathbb{E}_{\mathcal{P},\pi}[r_t|s_t]$. However, we actually cannot observe the inputs but only noisy samples $\gamma\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t$. The least-squares regression model does not account for this noise in the input variables, known as *error-in-variables* situation (Bradtke and Barto, 1996). As we will discuss in Section 2.2.1, this deficiency requires that two independent samples of s_{t+1} need to be drawn when being in state s_t, also known as the *double-sampling problem* (Baird, 1995). Hence, samples generated by a single roll-out cannot be used directly without introducing a bias. This bias corresponds to minimizing the *mean squared temporal-difference error* (Maei, 2011)

$$\text{MSTDE}(\boldsymbol{\theta}) = \mathbb{E}_{d,\mathcal{P},\pi}[\delta_t^2] = \mathbb{E}_d[\mathbb{E}_{\mathcal{P},\pi}[\delta_t^2|s_t]]. \tag{8}$$

The square is now inside the expectation. This cost function has a different optimum than the MSBE and, hence, minimizing it results in a different value function estimate.

Another possibility to avoid the optimization problems connected to the MSBE is instead to minimize the distance of the projected Bellman operator, also called the *mean squared projected Bellman error* (MSPBE). The Bellman operator in Equation (3) is independent of the feature representation, and, hence, $T\boldsymbol{V}_\theta$ may not be representable using the given features. The MSPBE only yields the error which is representable with the given features

and neglects the error orthogonal to the feature representation. Most prominent temporal-difference methods such as TD learning (Sutton, 1988), LSTD (Bradtke and Barto, 1996), GTD (Sutton et al., 2008) and TDC (Sutton et al., 2009) either directly minimize the MSPBE or converge to the same fixpoint (Tsitsiklis and van Roy, 1997; Sutton et al., 2008, 2009). The MSPBE is given by the squared distance between $\boldsymbol{V_\theta}$ and the representable function $\Pi T \boldsymbol{V_\theta}$ that is closest to $T\boldsymbol{V_\theta}$

$$\text{MSPBE}(\boldsymbol{\theta}) = \|\boldsymbol{V_\theta} - \Pi T \boldsymbol{V_\theta}\|^2_{\boldsymbol{D}}, \tag{9}$$

where Π is a projection operator which projects arbitrary value functions onto the space of representable functions $\mathcal{H}_{\boldsymbol{\phi}}$. For linear function approximation, the projection Π has the closed form

$$\Pi \boldsymbol{V} = \min_{V_\theta \in \mathcal{H}_\phi} \|\boldsymbol{V_\theta} - \boldsymbol{V}\|^2_{\boldsymbol{D}} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T D \boldsymbol{V},$$

where $\boldsymbol{\Phi} = [\boldsymbol{\phi}(s^1), \boldsymbol{\phi}(s^2), \dots \boldsymbol{\phi}(s^m)]^T \in \mathbb{R}^{m \times n}$ is the feature matrix consisting of rows with features of every state.

An illustration of the differences between the cost function can be found in Figure 4. The MSBE compares a parameterized value function $\boldsymbol{V_\theta}$ against $T\boldsymbol{V_\theta}$ (see the dotted distance in Figure 4), which may lie outside the space of parameterized functions $\mathcal{H}_{\boldsymbol{\phi}}$. The MSPBE first projects $T\boldsymbol{V_\theta}$ on the set of representable functions and, subsequently, calculates the error (solid distance).

Example 1 *For a simple example of the different distance functions, consider an MDP of two states and one action. The transition probabilities of the MDP and policy are uniform, that is,*

$$\boldsymbol{P}^\pi = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

The agent receives reward $r^1 = -0.8$ in the first state and $r^2 = 1.2$ in the second state. With a discount factor of $\gamma = 0.8$ the true value function is then given by $\boldsymbol{V}^\pi = (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi)^{-1}[-0.8\ 1.2]^T = [0\ 2]^T$. If we only use a single constant feature $\phi(s) = 1, \forall s \in \mathcal{S}$, the feature matrix is $\Phi = [1\ 1]^T$ and the parametrization $\boldsymbol{V_\theta} = [1\ 1]^T \boldsymbol{\theta}$ assigns the same value to all states. Hence, the true value function $\boldsymbol{V}^\pi$ cannot be represented by any parameter, that is, $\text{MSE}(\boldsymbol{\theta}) > 0\ \forall \boldsymbol{\theta}$. In addition, $\gamma \boldsymbol{P}^\pi \boldsymbol{V}_\theta$ is always a vector with equal components and subsequently $T\boldsymbol{V}_\theta = [-0.8\ 1.2]^T + \gamma \boldsymbol{P}^\pi \boldsymbol{V}_\theta$ has entries different from each other and the MSBE is always greater 0. One can easily verify that the projection is a simple average-operator $\boldsymbol{\Pi} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and that $\boldsymbol{\theta} = 1$ satisfies $\boldsymbol{V}_\theta - \Pi T \boldsymbol{V_\theta} = 0$, that is, $\text{MSPBE}(\boldsymbol{\theta}) = 0$.

The MSPBE circumvents the optimization problems connected to the MSBE, but instead loses the direct connection to the original MSE, the quantity we truly want to minimize. As shown by Sutton et al. (2009), the MSPBE can also be written as

$$\text{MSPBE}(\boldsymbol{\theta}) = \|\boldsymbol{V_\theta} - T\boldsymbol{V_\theta}\|^2_{\boldsymbol{U}} = \|\Phi^T \boldsymbol{D}(\boldsymbol{V_\theta} - T\boldsymbol{V_\theta})\|^2_{(\Phi^T \boldsymbol{D} \Phi)^{-1}}, \tag{10}$$

with $\boldsymbol{U} = \boldsymbol{D}\boldsymbol{\Phi}(\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D}$. A derivation of this formulation is given in Appendix A. This formulation reveals two important insights for understanding the MSPBE. First, the

MSPBE still measures the MSBE, just the metric is now defined as $\boldsymbol{U}$ instead of $\boldsymbol{D}$. Second, the minimum of the MSPBE is reached if and only if

$$\boldsymbol{\Phi}^T \boldsymbol{D}(\boldsymbol{V}_{\boldsymbol{\theta}} - T\boldsymbol{V}_{\boldsymbol{\theta}}) = \mathbb{E}_{d,\mathcal{P},\pi}[\delta_t \boldsymbol{\phi}_t] = \mathbf{0}. \tag{11}$$

This condition means that there is no correlation between the temporal-difference error δ_t and the feature vector $\boldsymbol{\phi}(s_t)$. Many algorithms, such as TD learning and LSTD have been shown to minimize the MSPBE as their fixpoints satisfy $\mathbb{E}_{d,\mathcal{P},\pi}[\delta_t \boldsymbol{\phi}_t] = \mathbf{0}$.

The insight that the fixpoint of TD learning has the property $\mathbb{E}_{d,\mathcal{P},\pi}[\delta_t \boldsymbol{\phi}_t] = \mathbf{0}$ has also motivated the *norm of the expected TD update*

$$\mathrm{NEU}(\boldsymbol{\theta}) = \|\boldsymbol{\Phi}^T \boldsymbol{D}(\boldsymbol{V}_{\boldsymbol{\theta}} - T\boldsymbol{V}_{\boldsymbol{\theta}})\|_2^2 = \mathbb{E}_{d,\mathcal{P},\pi}[\delta_t \boldsymbol{\phi}_t]^T \mathbb{E}_{d,\mathcal{P},\pi}[\delta_t \boldsymbol{\phi}_t] \tag{12}$$

as an alternative objective function. It shares the same minimum as the MSPBE but has a different shape, and therefore yields different optimization properties such as speed of convergence.

Many algorithms solve the problem of finding the minimum of the MSPBE indirectly by solving a nested optimization problem (Antos et al., 2008; Farahmand et al., 2008) consisting of the minimization of the *operator error (OPE)* and the *fixed-point error (FPE)*. The problem is given by

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}'} \mathrm{OPE}(\boldsymbol{\theta}', \boldsymbol{\omega}) = \arg\min_{\boldsymbol{\theta}'} \|\boldsymbol{V}_{\boldsymbol{\theta}'} - T\boldsymbol{V}_{\boldsymbol{\omega}}\|_{\boldsymbol{D}}^2 \quad \text{and} \tag{13}$$

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}'} \mathrm{FPE}(\boldsymbol{\theta}, \boldsymbol{\omega}') = \arg\min_{\boldsymbol{\omega}'} \|\boldsymbol{V}_{\boldsymbol{\theta}} - \boldsymbol{V}_{\boldsymbol{\omega}'}\|_{\boldsymbol{D}}^2 = \arg\min_{\boldsymbol{\omega}'} \|\boldsymbol{\Phi}(\boldsymbol{\theta} - \boldsymbol{\omega}')\|_{\boldsymbol{D}}^2. \tag{14}$$

Minimizing the MSPBE is split into two problems where we maintain two estimates of the parameters $\boldsymbol{\omega}$ and $\boldsymbol{\theta}$. In the operator problem, we try to approximate the Bellman operator applied to the value function $V_{\boldsymbol{\omega}}$ with $V_{\boldsymbol{\theta}}$. In the fixpoint problem, we reduce the distance between both parameter estimates $\boldsymbol{\omega}$ and $\boldsymbol{\theta}$. Many algorithms solve this problem by alternating between improving the operator and fixed-point error.

To see that the FPE-OPE solution indeed minimizes the MSPBE, we first look at the optimality conditions of the error functions. By considering the first order optimality criterion of the OPE

$$\mathbf{0} = \nabla_{\boldsymbol{\theta}} \mathrm{OPE}(\boldsymbol{\theta}, \boldsymbol{\omega}) = \boldsymbol{\Phi}^T \boldsymbol{D}(\boldsymbol{\Phi}\boldsymbol{\theta} - \gamma \boldsymbol{\Phi}'\boldsymbol{\omega} - \boldsymbol{R}) \underset{\boldsymbol{\omega}=\boldsymbol{\theta}}{=} \boldsymbol{\Phi}^T \boldsymbol{D}(\boldsymbol{V}_{\boldsymbol{\theta}} - T\boldsymbol{V}_{\boldsymbol{\theta}})$$

and using optimality in the fixpoint problem ($\boldsymbol{\omega} = \boldsymbol{\theta}$), we see that solving the nested OPE-FPE problem (13) – (14) indeed corresponds to minimizing the MSPBE from Equation (11). Note that the optimal value of the operator error is equal to the MSBE value due to $\boldsymbol{\omega} = \boldsymbol{\theta}$ and $\mathrm{OPE}(\boldsymbol{\omega}, \boldsymbol{\omega}) = \|\boldsymbol{V}_{\boldsymbol{\omega}} - T\boldsymbol{V}_{\boldsymbol{\omega}}\|_{\boldsymbol{D}}^2 = \mathrm{MSBE}(\boldsymbol{\omega})$. Yet, the problem does not corresponds to minimizing the MSBE as only one of the parameter vectors can change at a time. The OPE-FPE formulation is particularly appealing as it does not suffer from the double-sampling problem.

2.1.1 Fixpoint Discussion

Most temporal-difference methods for value estimation converge either to the minimum of MSBE or MSPBE. Thus, the properties of both functions as well as their relation to each

other and the mean squared error have been examined thoroughly by Schoknecht (2002) and Scherrer (2010) and in parts by Bach and Moulines (2011), Lazaric et al. (2010), Sutton et al. (2009) and Li (2008).

In the following, we summarize the most important results for both cost functions. First, MSBE and MSPBE are quadratic functions that are strongly convex if the features are linearly independent, that is, $\text{rank}(\boldsymbol{\Phi}) = m$. Linearly independent features are a necessary assumption for the convergence of most temporal-difference methods. Convexity of the cost function guarantees that optimization techniques such as gradient descent do not get stuck in a non-global minimum. Second, the MSBE is larger than the MSPBE for any fixed $\boldsymbol{\theta}$ as

$$\text{MSBE}(\boldsymbol{\theta}) = \text{MSPBE}(\boldsymbol{\theta}) + \|TV_{\boldsymbol{\theta}} - \Pi TV_{\boldsymbol{\theta}}\|^2_{\boldsymbol{D}},$$

where $\|TV_{\boldsymbol{\theta}} - \Pi TV_{\boldsymbol{\theta}}\|^2_{\boldsymbol{D}}$ is the projection error (dashed distance in Figure 4). As Π is an orthogonal projection, this insight follows directly from the Pythagorean theorem (cf. Figure 4).

In addition, Williams and Baird (1993) as well as Scherrer (2010) have derived a bound on the MSE by the MSBE

$$\text{MSE}(\boldsymbol{\theta}) \leq \frac{\sqrt{C(d)}}{1-\gamma} \text{MSBE}(\boldsymbol{\theta}), \text{ with } C(d) = \max_{s^i, s^j} \frac{\sum_a P(s^j|s^i, a)\pi(a|s^i)}{d^\pi(s^i)} \tag{15}$$

where $C(d)$ is a constant concentration coefficient depending on π and $\mathcal{P}$. The numerator contains the average probability of transitioning from s^i to s^j while the denominator is the probability of the stationary distribution at state s^i. The term $C(d)$ becomes minimal if the transitions of the MDP are uniform. For the MSPBE no such general bound exists, as the MSPBE only considers part of the MSBE and ignores the projection error. Under mild conditions, the optimal value of MSPBE is always 0, while optima of MSE and MSBE may have larger values (see Example 1). Bertsekas and Tsitsiklis (1996) and Scherrer (2010) provide an example, where the projection error is arbitrarily large, and the MSE value of the MSPBE optimum is therefore unbounded as well.

MSBE and MSPBE solutions (also referred to as fixpoints, as they satisfy $\boldsymbol{V}_{\boldsymbol{\theta}} = T\boldsymbol{V}_{\boldsymbol{\theta}}$ and $\boldsymbol{V}_{\boldsymbol{\theta}} = \Pi T\boldsymbol{V}_{\boldsymbol{\theta}}$ respectively) have been characterized as different projections of the true value function V onto the space of representable functions $\mathcal{H}_\phi$ by Schoknecht (2002) and Scherrer (2010). These results imply that, if $V^\pi \in \mathcal{H}_\phi$, that is, there exist parameters for the true value function, algorithms optimizing MSPBE or MSBE converge to the true solution. If V^π cannot be represented, the optima of MSBE and MSPBE are different in general. The natural question, which objective yield better solutions in terms of MSE value, is addressed by Scherrer (2010). The minimum of MSPBE often has a lower mean squared error, however, the solution may get unstable and may yield estimates arbitrarily far away from the true solution V^π. Scherrer (2010) illustrated this effect by an example MDP with unstable MSPBE solutions for certain settings of the discount factor γ. On the other hand, the MSBE has been observed to have higher variance in its estimate and is therefore harder to minimize than the MSPBE even if we solve the double-sampling problem (see Section 3.2). While the bound in Equation (15) gives a quality guarantee for the MSBE solution, in practice, it may be too loose for many MDPs as shown in Section 3. In addition, the MSPBE was observed to result in control policies of higher quality, if used as objective

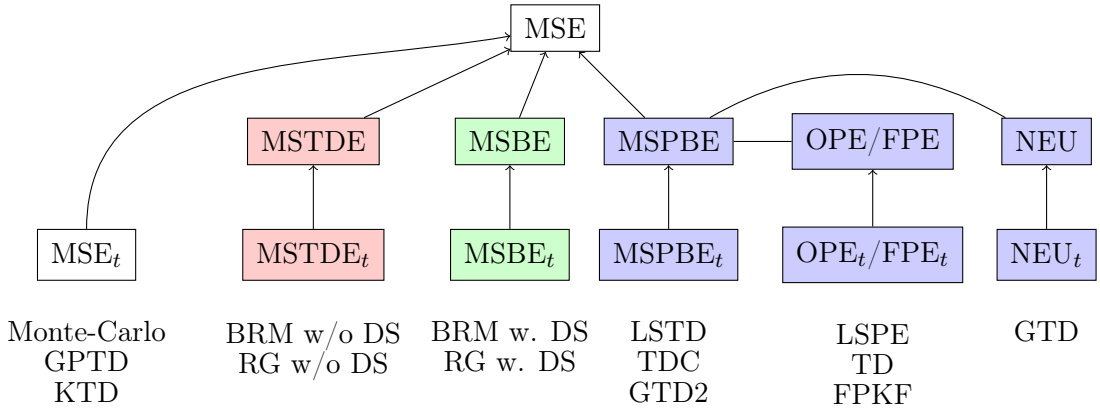


Figure 5: Relations between cost functions and temporal-difference learning algorithms. The methods are listed below the sample-based objective function, which they minimize at timestep t, denoted by the subscript t. The basic idea of temporal difference learning is to optimize for a different (potentially biased) objective function instead of the MSE directly, since its sample-based approximation MSE_t at timestep t converges very slowly to the MSE (due to the large sample variance). The MSPBE, OPE/FPE and NEU objectives (blue shaded) share the same fixed-point and their algorithms converge therefore to the same solution, but possibly at different pace.

functions in a policy iteration loop (Lagoudakis and Parr, 2003). For these reasons the MSPBE is typically preferred. While MSBE and MSPBE have been studied in detail, the quality of the MSTDE cost function and its exact relation to the MSBE are still open questions.

Figure 5 provides a visual overview of the important cost functions and their respective algorithms which are introduced in the following section.

2.2 Algorithm Design

In the following discussion, we will categorize temporal-difference methods as a combination of cost functions and optimization techniques. While we introduced the former in the previous section, we now focus on the optimization techniques. Temporal-difference methods for value function estimation rely either on gradient-based approaches, least-squares minimization techniques or probabilistic models to minimize the respective cost function. Each optimization approach and the consequent family of temporal-difference methods is presented in Sections 2.2.1, 2.2.2 and 2.2.3. We do not give the complete derivation for every method but instead aim for providing their key ingredients and highlighting similarities and difference between the families. Table 1 lists all algorithms presented in this section along with their most important properties.

	Fixpoint	Runtime Complexity	Eligibility Traces	Off-Policy Convergence	Idea
TD	MSPBE	$O(n)$	TD(λ)	no	bootstrapped SGD of MSE
GTD	MSPBE	$O(n)$	-	yes	SGD of NEU
GTD2	MSPBE	$O(n)$	GTD2(λ)	yes	SGD of MSPBE
TDC	MSPBE	$O(n)$	GTD(λ)/TDC(λ)	yes	SGD of MSPBE
RG	MSBE / MSTDE	$O(n)$	gBRM(λ)	yes	SGD of MSBE
BRM	MSBE / MSTDE	$O(n^2)$	BRM(λ)	yes	∇MSBE $= 0$
LSTD	MSPBE	$O(n^2)$	LSTD(λ)	yes	∇MSPBE $= 0$
LSPE	MSPBE	$O(n^2)$	LSPE(λ)	yes	recursive LS Min.
FPKF	MSPBE	$O(n^2)$	FPKF(λ)	?	recursive LS Min.
KTD	MSE	$O(n^2)$	-	no	Parameter Tracking by Kalman Filtering
GPTD	MSE	$O(n^2)$	GPTD(λ)	no	Gaussian Process on V

Table 1: Overview of Temporal-Difference Methods. The methods can divided into gradient-based approaches, least-squares methods and probabilistic models (from top to bottom, separated by horizontal lines). The prior beliefs in probabilistic models acts as a regularization of the cost function. The fixpoint of the residual-gradient algorithm (RG) and Bellman residual minimization (BRM) depends on whether independent second samples for successor states are used or not. The convergence analysis of FPKF for off-policy estimation is still an open problem (Scherrer and Geist, 2011; Geist and Scherrer, 2013).

2.2.1 Gradient-Based Approaches

One family of temporal-difference methods relies on *stochastic gradient descent (SGD)* to optimize their cost function. This optimization technique is directly based on stochastic approximation going back to Robbins and Monro (1951).

Stochastic gradient descent is typically applied to functions of the form $f(\boldsymbol{\theta}) = \mathbb{E}_{p(x)}[g(x;\boldsymbol{\theta})]$, where the expectation is usually approximated by samples and the distribution $p(x)$ is independent of $\boldsymbol{\theta}$. The parameter update in gradient descent follows the negative gradient, that is,

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla f(\boldsymbol{\theta}_k) = \boldsymbol{\theta}_k - \alpha_k \mathbb{E}_{p(x)}[\nabla g(x;\boldsymbol{\theta}_k)],$$

where α_k denotes a step-size. While in ordinary gradient descent, also denoted as batch gradient descent, the gradient is calculated using all samples, stochastic gradient descent only evaluates the gradient for one sample $\tilde{x}$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla g(\tilde{x}; \boldsymbol{\theta}_k) \quad \text{with} \quad \tilde{x} \sim p(x).$$

Stochastic updates are guaranteed to converge to a local minimum of f under the mild stochastic approximation conditions (Robbins and Monro, 1951) such that the step-sizes $\alpha_k \geq 0$ satisfy

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

All cost functions in Section 2.1 are expectations with respect to the stationary state distribution d^π. Additionally, observations arrive sequentially in online learning, and therefore, stochastic gradient descent is particularly appealing as it requires only one sample per update. Stochastic gradient temporal-difference methods update the parameter estimate $\boldsymbol{\theta}_t$ after each observed transition from the current state s_t to the next state s_{t+1} with action a_t and reward r_t.

Temporal-Difference Learning. Learning signals similar to temporal differences have been used before, for example by Samuel (1959), but the general concept was first introduced by Sutton (1988) with the *temporal-difference (TD) learning* algorithm. It is considered to be the first use of temporal differences for value-function estimation. Sometimes, also subsequent approaches are referred to as TD learning algorithms. To avoid ambiguity, we use the term TD learning only for the first algorithm presented by Sutton (1988) and denote all other approaches with temporal-difference methods.

The idea behind TD learning is to minimize the MSE where we use $T V_{\boldsymbol{\theta}_t}$ as approximation for the true value function V^π. Minimizing this function $\|\boldsymbol{V}_{\boldsymbol{\theta}} - T\boldsymbol{V}_{\boldsymbol{\theta}_t}\|^2_{\boldsymbol{D}}$ w.r.t. $\boldsymbol{\theta}$ by stochastic gradient descent yields the update rule of TD learning in its basic form

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t [r_t + \gamma V_{\boldsymbol{\theta}_t}(s_{t+1}) - V_{\boldsymbol{\theta}_t}(s_t)] \boldsymbol{\phi}_t = \boldsymbol{\theta}_t + \alpha_t \delta_t \boldsymbol{\phi}_t. \tag{16}$$

As the target values $T\boldsymbol{V}_{\boldsymbol{\theta}_t}$ change over time ($T\boldsymbol{V}_{\boldsymbol{\theta}_0}$,$T\boldsymbol{V}_{\boldsymbol{\theta}_1}$,$T\boldsymbol{V}_{\boldsymbol{\theta}_2}$,...), TD learning does not perform stochastic gradient descent on a well-defined objective function. Thus, general stochastic approximation results are not applicable and in fact several issues emerge from the function change. TD learning as in Equation (16) is only guaranteed to converge if the stationary state distribution d^π is used as sampling distribution, that is, on-policy estimation. If the value function is estimated from off-policy samples, we can easily construct scenarios where TD learning diverges (Baird, 1995). For a more detailed discussion of the off-policy case we refer to Section 2.4.2. In addition, Tsitsiklis and van Roy (1997) have shown that the TD learning algorithm can diverge for non-linear function approximation.

TD learning can be understood more clearly as minimization of the nested optimization problem introduced in Equations (13) and (14). More precisely, TD learning first optimizes the fixpoint problem from Equation (14) by setting $\boldsymbol{\omega} = \boldsymbol{\theta}_t$ and then performs a stochastic gradient step on the operator problem from Equation (13). Hence, we can already conclude that the convergence point of TD learning is given by

$$\boldsymbol{V}_{\boldsymbol{\theta}} = \Pi T^\pi \boldsymbol{V}_{\boldsymbol{\theta}},$$

which is the minimum of the MSPBE objective in Equation (9).

As the results in Section 3 show, the performance of TD learning depends on good step-sizes α_t. Hutter and Legg (2007) aim at overcoming the need for optimizing this hyper-parameter. They re-derived TD learning by formulating the least-squares value-function estimate as an incremental update, which yielded automatically adapting learning rates for tabular feature representations. Dabney and Barto (2012) extended this approach to arbitrary features and additionally proposed another adapting step-size scheme which ensures that the value estimates do not increase the temporal-difference errors $\delta_0, \delta_1, \ldots, \delta_t$ observed in previous timesteps. Autostep (Mahmood et al., 2012), a learning-rate adaptation approach for incremental learning algorithms based on stochastic gradient, yields individual step-sizes for each feature which may boost learning speed. It relies on a meta-step-size but works well for a wide range of step lengths which makes specifying the meta-parameter easier than the step-size for TD learning directly.

Residual-Gradient Algorithm. The *residual-gradient* (RG) algorithm (Baird, 1995) minimizes the *mean squared Bellman error (MSBE)* directly by stochastic gradient descent. Its update rule in the most basic form is given by

$$\begin{aligned}\boldsymbol{\theta}_{t+1} =& \boldsymbol{\theta}_t + \alpha_t[r_t + \gamma V_{\boldsymbol{\theta}_t}(s_{t+1}) - V_{\boldsymbol{\theta}_t}(s_t)](\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}) \\ =& \boldsymbol{\theta}_t + \alpha_t\delta_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}).\end{aligned}$$

The difference compared to TD learning is that the gradient of $V_{\boldsymbol{\theta}_t}(s_{t+1})$ with respect to $\boldsymbol{\theta}_t$ is also incorporated into the update.

Unfortunately, RG methods suffer from the double-sampling problem mentioned in Section 2.1. Consider the gradient of the MSBE, in Equation (5), given by

$$2\mathbb{E}_d\Big[\Big(V_{\boldsymbol{\theta}}(s) - \mathbb{E}_{\pi,\mathcal{P}}[r(s_t,a_t) + \gamma\phi(s_{t+1})^T\boldsymbol{\theta}|s_t = s]\Big)\Big(\phi(s) - \gamma\mathbb{E}_{\pi,\mathcal{P}}[\phi(s_{t+1})|s_t = s]\Big)\Big]. \quad (17)$$

The outer expectation is computed over the steady state distribution and can be replaced by a single term in stochastic gradient. Both inner expectations are taken over the joint of the policy π and the transition distribution $\mathcal{P}$ of the MDP. Multiplying out the brackets yields $\gamma^2\mathbb{E}[\boldsymbol{\phi}_{t+1}]\mathbb{E}[\boldsymbol{\phi}_{t+1}]^T\boldsymbol{\theta}$ besides other terms. If we replace both expectations with the current observation $\boldsymbol{\phi}_{t+1}$, we obtain a biased estimator since

$$\boldsymbol{\phi}_{t+1}\boldsymbol{\phi}_{t+1}^T \underset{\text{Stoch. Approx.}}{\approx} \mathbb{E}[\boldsymbol{\phi}_{t+1}\boldsymbol{\phi}_{t+1}^T|s_t] = \mathbb{E}[\boldsymbol{\phi}_{t+1}|s_t]\mathbb{E}[\boldsymbol{\phi}_{t+1}|s_t]^T + \mathrm{Cov}[\boldsymbol{\phi}_{t+1},\boldsymbol{\phi}_{t+1}].$$

Hence, updating the parameters only with the current sample is biased by the covariances $\mathrm{Cov}[\boldsymbol{\phi}_{t+1},\boldsymbol{\phi}_{t+1}]$ and $\mathrm{Cov}[r_t,\boldsymbol{\phi}_{t+1}]$ with respect to $\mathcal{P}$ and π. While this effect can be neglected for deterministic MDPs and policies (since $\mathrm{Cov}[\boldsymbol{\phi}_{t+1},\boldsymbol{\phi}_{t+1}] = 0, \mathrm{Cov}[r_t,\boldsymbol{\phi}_{t+1}] = 0$), the residual-gradient algorithm does not converge to a minimizer of the MSBE for stochastic MDPs. It has been shown by Maei (2011) that the residual-gradient algorithm converges to a fixed point of the *mean squared TD error*[2] defined in Equation (8) instead. Alternatively,

2. A different characterization of the RG fixpoint was derived by Schoknecht (2002).

each inner expectation in Equation (17) can be replaced by independently drawn samples a'_t, r'_t, s'_{t+1} and a''_t, r''_t, s''_{t+1} of the transition

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t[r'_t + \gamma V_{\boldsymbol{\theta}_t}(s'_{t+1}) - V_{\boldsymbol{\theta}_t}(s_t)](\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}''_{t+1}).$$

With double samples, the residual-gradient algorithm indeed convergences to a fixpoint of the MSBE. However, a second sample is only available, if the model of the MDPs is known or a previously observed transition from the same state is reused. While there have been efforts to avoid the double-sampling problem for other approaches such as the projected fixpoint methods or Bellman residual minimization (Farahmand et al., 2008), it is still an open question whether the bias of the residual-gradient algorithm with single samples can be removed by similar techniques.

Projected-Fixpoint Methods. The key idea of the projected fixpoint algorithms is to minimize the MSPBE directly by stochastic gradient descent (Sutton et al., 2009) and, therefore, overcome the issue of TD learning which alters the objective function between descent steps.

Sutton et al. (2009) proposed two different stochastic gradient descent techniques. The derivation starts by writing the MSPBE in a different form given by

$$\text{MSPBE}(\boldsymbol{\theta}) = \mathbb{E}[\delta_t\boldsymbol{\phi}_t]^T\mathbb{E}[\boldsymbol{\phi}_t\boldsymbol{\phi}_t^T]^{-1}\mathbb{E}[\delta_t\boldsymbol{\phi}_t]. \tag{18}$$

The proof of this equation is provided in Appendix A, Equation (43). We can write the gradient of Equation (18) as

$$\nabla\,\text{MSPBE}(\boldsymbol{\theta}) = -2\mathbb{E}[(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})\boldsymbol{\phi}_t^T]\mathbb{E}[\boldsymbol{\phi}_t\boldsymbol{\phi}_t^T]^{-1}\mathbb{E}[\delta_t\boldsymbol{\phi}_t] \tag{19}$$

$$= -2\mathbb{E}[\delta_t\boldsymbol{\phi}_t] + 2\gamma\mathbb{E}[\boldsymbol{\phi}_{t+1}\boldsymbol{\phi}_t^T]\mathbb{E}[\boldsymbol{\phi}_t\boldsymbol{\phi}_t^T]^{-1}\mathbb{E}[\delta_t\boldsymbol{\phi}_t]. \tag{20}$$

The gradient contains a product of expectations of $\boldsymbol{\phi}_{t+1}$ and δ_t in both forms (Equation 19 and 20). As both terms depend on the transition distribution of the MDP, minimizing Equation (18) with stochastic gradient descent again requires two independently drawn samples, as in the residual-gradient algorithm. To circumvent this limitation, a long-term quasi-stationary estimate $\boldsymbol{w}$ of

$$\mathbb{E}[\boldsymbol{\phi}_t\boldsymbol{\phi}_t^T]^{-1}\mathbb{E}[\delta_t\boldsymbol{\phi}_t] = \boldsymbol{(\Phi^T D\Phi)^{-1}\Phi^T D}(T\boldsymbol{V_\theta} - \boldsymbol{V_\theta}) \tag{21}$$

is calculated. To obtain an iterative update for $\boldsymbol{w}$, we realize that the right side of Equation (21) is the solution to the following least-squares problem

$$J(\boldsymbol{w}) = \|\boldsymbol{\Phi}^T\boldsymbol{w} - (T\boldsymbol{V}_\theta - \boldsymbol{V}_\theta)\|_2^2.$$

This least-squares problem can also be solved by stochastic gradient descent with the update rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \beta_t(\delta_t - \boldsymbol{\phi}_t^T\boldsymbol{w}_t)\boldsymbol{\phi}_t,$$

and the step-size β_t. Inserting the estimate $\boldsymbol{w}_t$ into Equation (19) and Equation (20) allows us to rewrite the gradient with a single expectation

$$\nabla\,\text{MSPBE}(\boldsymbol{\theta}) = -2\mathbb{E}[(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})\boldsymbol{\phi}_t]^T\boldsymbol{w}_t \tag{22}$$

$$= -2\mathbb{E}[\delta_t\boldsymbol{\phi}_t] + 2\gamma\mathbb{E}[\boldsymbol{\phi}_{t+1}\boldsymbol{\phi}_t^T]\boldsymbol{w}_t. \tag{23}$$

Minimizing with the gradient of the form of Equation (22) is called the *GTD2 (gradient temporal-difference learning 2)* algorithm with update rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})\boldsymbol{\phi}_t^T\boldsymbol{w}_t,$$

and using the form of Equation (23) yields the *TDC (temporal-difference learning with gradient correction)* algorithm (Sutton et al., 2009)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t(\delta_t\boldsymbol{\phi}_t - \gamma(\boldsymbol{\phi}_t^T\boldsymbol{w}_t)\boldsymbol{\phi}_{t+1}).$$

As $\boldsymbol{\theta}$ and $\boldsymbol{w}$ are updated at the same time, the choice of step-sizes α_t and β_t are critical for convergence (see also our experiments in Section. 3). Both methods can be understood as a nested version of stochastic gradient descent optimization. TDC is similar to TD learning, but with an additional term to adjust the TD update to approximate the real gradient of MSPBE. The right side of Figure 7 shows this corrections and compares both stochastic approximations to descent following the true gradient. Both algorithms minimize the MSPBE but show different speeds of convergence, as we will also illustrate in the discussion of experimental results in Section 3.

The predecessor of the GTD2 algorithm is the GTD algorithm (Sutton et al., 2008). It minimizes the NEU cost function from Equation (12) by stochastic gradient descent. The gradient of NEU is given by

$$\nabla \mathrm{NEU}(\theta) = -2\mathbb{E}[(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})\boldsymbol{\phi}_t^T]\mathbb{E}[\delta_t\boldsymbol{\phi}_t].$$

One of the two expectations needs to be estimated by a quasi-stationary estimate in analogy to the other projected fixpoint methods. Hence, the term $E[\delta_t\boldsymbol{\phi}_t]$ is replaced by $\boldsymbol{u}$ which is updated incrementally by

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \beta_t(\boldsymbol{\phi}_t^T\delta_t - \boldsymbol{u}_t) = (1-\beta_t)\boldsymbol{u}_t + \beta_t\boldsymbol{\phi}_t^T\delta_t.$$

The updates for $\boldsymbol{\theta}$ of GTD are then given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})\boldsymbol{\phi}_t^T\boldsymbol{u}_t.$$

The update rule for GTD is similar to GTD2 but the quasi stationary estimates $\boldsymbol{w}$ and $\boldsymbol{u}$ are different. While GTD2 searches for the best linear approximation $\boldsymbol{\phi}_t^T\boldsymbol{w}$ of $\mathbb{E}[\delta_t]$, GTD tries to approximate $\mathbb{E}[\delta_t\boldsymbol{\phi}_t]$ with $\boldsymbol{u}$. As shown by Sutton et al. (2009) and our experiments, GTD2 converges faster and should be preferred over GTD.

All gradient-based temporal-difference methods only require sums and products of vectors of length n to update the parameters. Thus, they run in $O(n)$ time per update. The initial value of $\boldsymbol{\theta}$ has a tremendous influence on the convergence speed of gradient-based methods. While other approaches such as LSTD do not require an initial values, the gradient based approaches can benefit from good parameter guesses, which are available in numerous applications. For example, the parameter vector learned in previous steps of policy iteration can be used as initial guesses to speed up learning. Fixed-Point Kalman Filtering (FPKF) proposed by Choi and Roy (2006) is a descent method, that has a close relationship to TD learning. Yet, as it is motivated by least-squares minimization, we present it in the next section.

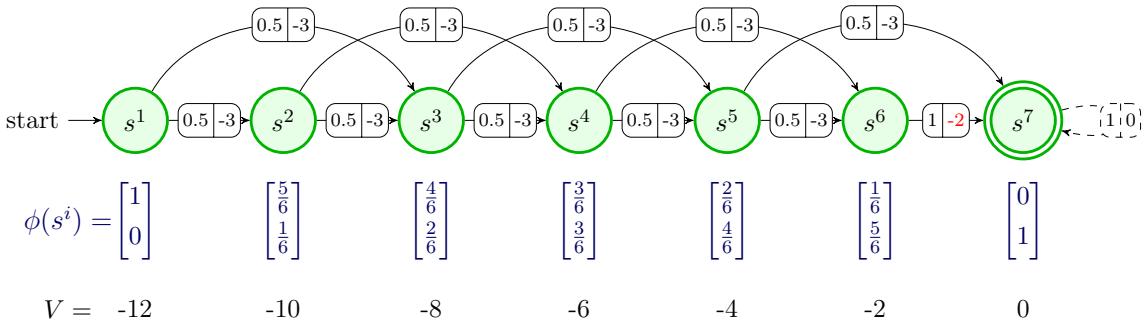


Figure 6: 7-State Boyan Chain MDP from Boyan (2002). Each transition is specified by a probability (left number) and a reward (right number). There are no actions to chose for the agent. The features are linearly increasing / decreasing from left to right and are capable of representing the true value function with parameters $\boldsymbol{\theta} = [-12, 0]^T$.

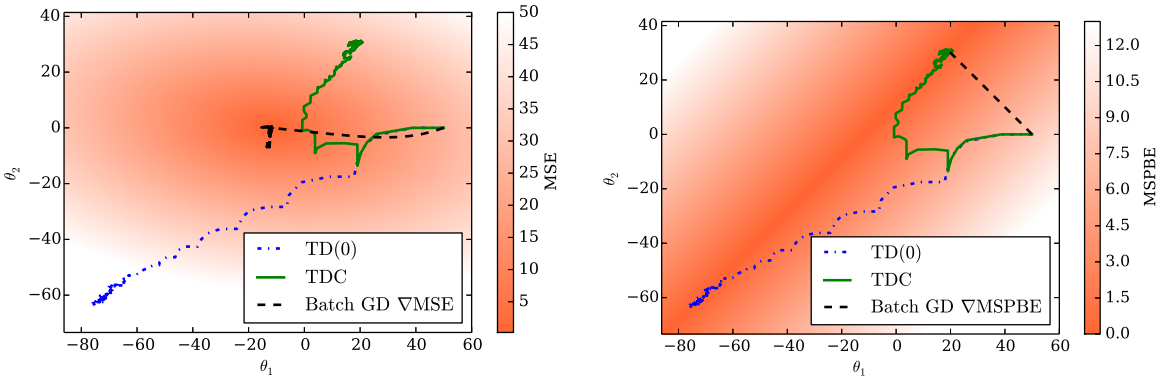


Figure 7: Comparison of Gradient Descent and TD learning for the MDP of Figure 6 with $\gamma = 1$: The left plot shows the mean squared error. Ideally, we want to find parameters that minimize this objective, however, TD(0) and TDC only find a minimum of the MSPBE (Eq. 9, right plot). The dashed lines show the parameter iterates of batch gradient descent of the respective cost functions. Stochastic gradient methods such as TD and TDC are slower since they can only update $\boldsymbol{\theta}_1$ for samples from the beginning of an episode and $\boldsymbol{\theta}_2$ for samples from the end (cf. the features from Figure 6). Comparison of both plots shows, that a fixpoint of MSPBE can give arbitrarily bad results for MSE, as the problem is not discounted and the guarantees of Section 2.1 do not hold. The MSPBE and MSBE measures only compare the difference of the value of a state and its successor's value. Hence, all parameters which yield differences of 2 and arbitrary value of the terminal state are optimal.

2.2.2 Least-Squares Approaches

Least-squares approaches use all previously observed transitions to determine either the value function directly in one step or an update of the value function. All considered objective functions (cf. Section 2.1) have the form of a standard linear regression problem

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_{\boldsymbol{U}}^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{U} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}),$$

with respect to the (semi-)norm induced by a positive semi-definite matrix $\boldsymbol{U} \in \mathbb{R}^{k \times k}$. While the targets are denoted by $\boldsymbol{y} \in \mathbb{R}^k$, $\boldsymbol{X} \in \mathbb{R}^{k \times n}$ is the matrix consisting of rows of basis vectors. Setting the gradient of the objective function with respect to $\boldsymbol{\theta}$ to 0 yields the closed-form least-squares solution

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T \boldsymbol{U} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{U} \boldsymbol{y}.$$

Least-Squares Temporal-Difference Learning. The most prominent least-squares method for policy evaluation is least-squares temporal-difference (LSTD) learning (Bradtke and Barto, 1996; Boyan, 2002). LSTD uses the MSPBE as objective function. The least-squares solution of the MSPBE from Equation (10) is given by

$$\boldsymbol{\theta} = \underbrace{(\boldsymbol{\Phi}^T \boldsymbol{D} (\boldsymbol{\Phi} - \gamma \boldsymbol{P} \boldsymbol{\Phi}))}_{\boldsymbol{A}}^{-1} \underbrace{\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{R}}_{\boldsymbol{b}}. \tag{24}$$

During the derivation of this solution (see Appendix A) many terms cancel out, including the ones which are connected to the double-sampling problem—in contrast to the analytical solution for minimizing the MSBE shown in Equation (31). Alternatively, LSTD can be derived by considering the analytical solution of the OPE problem from the OPE–FPE formulation (Equations 13 and 14) given by

$$\begin{aligned}\boldsymbol{\theta} =& (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} (\boldsymbol{R} + \gamma \boldsymbol{P} \boldsymbol{\Phi} \boldsymbol{\omega}) \\ \underset{\boldsymbol{\omega} = \boldsymbol{\theta}}{=}& (\boldsymbol{\Phi}^T \boldsymbol{D} (\boldsymbol{\Phi} - \gamma \boldsymbol{P} \boldsymbol{\Phi}))^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{R}.\end{aligned}$$

The LSTD solution in the second line is obtained by inserting the solution of the FPE problem, $\boldsymbol{\omega} = \boldsymbol{\theta}$, into the first line and re-ordering the terms.

LSTD explicitly estimates $\boldsymbol{A} = \boldsymbol{\Phi}^T \boldsymbol{D} (\boldsymbol{\Phi} - \gamma \boldsymbol{P} \boldsymbol{\Phi})$ and $\boldsymbol{b} = \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{R}$ and then determines $\boldsymbol{\theta} = \boldsymbol{A}^{-1} \boldsymbol{b}$ robustly (for example with singular value decomposition). The estimates $\boldsymbol{A}_t, \boldsymbol{b}_t$ at time t can be computed iteratively by

$$\boldsymbol{A}_{t+1} = \boldsymbol{A}_t + \boldsymbol{\phi}_t [\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}]^T, \tag{25}$$

$$\boldsymbol{b}_{t+1} = \boldsymbol{b}_t + \boldsymbol{\phi}_t r_t, \tag{26}$$

and converge to $\boldsymbol{A}, \boldsymbol{b}$ (Nedic and Bertsekas, 2003) for $t \to \infty$. For calculating $\boldsymbol{\theta}_t$ we need to invert a $n \times n$ matrix. This computational cost can be reduced from $O(n^3)$ to $O(n^2)$ by updating $\boldsymbol{A}_t^{-1}$ directly and maintaining an estimate $\boldsymbol{\theta}_t = \boldsymbol{A}_t^{-1} \boldsymbol{b}_t$ (Nedic and Bertsekas, 2003; Yu, 2010). The direct update of $\boldsymbol{A}_t^{-1}$ can be derived using the Sherman-Morrison formula

$$(\boldsymbol{A}_t + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \boldsymbol{A}_t^{-1} - \frac{\boldsymbol{A}_t^{-1} \boldsymbol{u} \boldsymbol{v}^T \boldsymbol{A}_t^{-1}}{1 + \boldsymbol{v}^T \boldsymbol{A}_t^{-1} \boldsymbol{u}} \tag{27}$$

with vectors $\boldsymbol{u} := \boldsymbol{\phi}_t$ and $\boldsymbol{v} := \boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}$. The resulting recursive LSTD algorithm is listed in Appendix C in a more general form with eligibility traces and off-policy weights (for the basic form set $\lambda = 0$ and $\rho_t = 1$).

An initial guess $\boldsymbol{A}_0^{-1}$ has to be chosen by hand. $\boldsymbol{A}_0^{-1}$ corresponds to the prior belief of $\boldsymbol{A}^{-1}$ and is ideally the inverse of the null-matrix. In practice, the choice $\boldsymbol{A}_0^{-1} = \epsilon\boldsymbol{I}$ with $\epsilon \gg 0$ works well. Small values for ϵ act as a regularizer on $\boldsymbol{\theta}$.

Interestingly, LSTD has a model-based reinforcement learning interpretation. For lookup table representations, the matrix $\boldsymbol{A}_t$ contains an empirical model of the transition probabilities. To see that, we write $\boldsymbol{A}_t$ and $\boldsymbol{b}_t$ as

$$\boldsymbol{A}_t = \boldsymbol{N} - \boldsymbol{C} = \boldsymbol{N}(\boldsymbol{I} - \gamma\hat{\boldsymbol{P}}), \quad \boldsymbol{b}_t = \boldsymbol{N}\hat{\boldsymbol{R}},$$

where $\boldsymbol{N}$ is a diagonal matrix containing the state visit counts up to time step t. The elements C_{ij} of matrix $\boldsymbol{C}$ contain the number of times a transition from state i to state j has been observed. The matrix $\hat{\boldsymbol{P}} = \gamma^{-1}\boldsymbol{N}^{-1}\boldsymbol{C}$ denotes the estimated transition probabilities and $\hat{\boldsymbol{R}}_i$ denotes the average observed reward when being in state i. Note that, in this case, the LSTD solution

$$\boldsymbol{\theta}^* = \left(\boldsymbol{N}\left(\boldsymbol{I} - \gamma\hat{\boldsymbol{P}}\right)\right)^{-1}\boldsymbol{N}\hat{\boldsymbol{R}} = \left(\boldsymbol{I} - \gamma\hat{\boldsymbol{P}}\right)^{-1}\hat{\boldsymbol{R}}$$

exactly corresponds to model based policy evaluation with an estimated model. For approximate feature spaces, the equivalence to model-based estimation is lost, but the intuition remains the same. A more detailed analysis of this connection can be found in the work of Boyan (2002) and Parr et al. (2008)

Least-Squares Policy Evaluation. The least-squares policy evaluation (LSPE) algorithm proposed by Nedic and Bertsekas (2003) shares similarities with TD learning and the LSTD method as it combines the idea of least-squares solutions and gradient descent steps. This procedure can again be formalized with the nested OPE-FPE problem from Equations (13) and (14). First, LSPE solves the operator problem

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{\Phi}\boldsymbol{\theta} - T\boldsymbol{\Phi}\boldsymbol{\omega}_t\|_{\boldsymbol{D}}^2 \tag{28}$$

in closed form with the least-squares solution. Then, it decreases the fixpoint error by performing a step in the direction of the new $\boldsymbol{\theta}_{t+1}$

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_t(\boldsymbol{\theta}_{t+1} - \boldsymbol{\omega}_t), \tag{29}$$

where $\alpha_t \in (0, 1]$ is a predefined step size. The vector $\boldsymbol{\omega}_t$ is the output of the algorithm at time-step t, that is, the parameter estimate for the value function. In practice, the step-sizes are large in comparison to stochastic gradient approaches and can often be set to 1.

The solution of the LSPE problem from Equation (28) is given by

$$\boldsymbol{\theta}_{t+1} = \underbrace{(\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Phi})^{-1}}_{\boldsymbol{M}}\boldsymbol{\Phi}^T\boldsymbol{D}(\boldsymbol{R} + \gamma\boldsymbol{\Phi}'\boldsymbol{\omega}_t), \tag{30}$$

where $\boldsymbol{\Phi}'$ is the matrix containing the features of the successor states, that is, $\boldsymbol{\Phi}' = \boldsymbol{P}^\pi \boldsymbol{\Phi}$. The current estimate $\boldsymbol{M}_t$ of $(\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1}$ can again be updated recursively with the Sherman-Morrison formula from Equation (27) similar to before, which yields the update rule of LSPE summarized in Algorithm 7 in Appendix C. As LSPE solves the nested OPE-FPE problem, it also converges to the MSPBE fixpoint (Nedic and Bertsekas, 2003). Hence, LSPE and LSTD find the same solution, but LSPE calculates the value function recursively using least-squares solutions of the OPE problem while LSTD acquires the value function directly by solving both, OPE and FPE, problems in closed form. LSPE allows adapting the step-sizes α_t of the updates and using prior knowledge for the initial estimate of $\boldsymbol{\omega}_0$, which serves as a form of regularization. Therefore, LSPE does not aim for the minimum of the MSPBE approximated by samples up to the current timestep, it instead refines the previous estimates. Such behavior may avoid numerical issues of LSTD and is less prone to over-fitting.

Fixed-Point Kalman Filtering. Kalman filtering is a well known second-order alternative to stochastic gradient descent. Choi and Roy (2006) applied the Kalman filter to temporal-difference learning, which resulted in the Fixed-Point Kalman Filtering (FPKF) algorithm.

As in TD learning, FPKF solves the nested optimization problem from Equations (13) and (14) and hence finds the minimum of the MSPBE objective function. However, instead of stochastic gradient descent, FPKF performs a second order update by multiplying the standard TD learning update with the inverse of the Hessian H_t of the operator error given in Equation (13). FPKF can therefore be also understood as an approximate Newton-method on the OPE problem. The Hessian H_t is given by the second derivative of the OPE

$$\boldsymbol{H}_t = \frac{1}{t} \sum_{i=1}^{t} \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T.$$

Note that the Hessian is calculated from the whole data set $i = 1, \dots, t$ up to the current time step and does not depend on the parameters $\boldsymbol{\theta}$. The update rule of FPKF is thus given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \boldsymbol{H}_t^{-1} \boldsymbol{\phi}_t \delta_t.$$

This update rule can also be derived directly from the Kalman filter updates if α_t is set to $1/t$. See Figure 8 for a graphical model illustrating the Kalman Filter assumptions (e.g., the definition of the evolution and observation function). For small values of t, the matrix $\boldsymbol{H}_t$ becomes singular. In this case $\boldsymbol{H}_t$ needs to be regularized or the pseudo-inverse of $\boldsymbol{H}_t$ has to be used. As FPKF is a second order method, it typically converges with fewer iterations than TD but comes with additional price of estimating $\boldsymbol{H}_t^{-1}$. Analogously to the derivation of LSPE, $\boldsymbol{H}_t^{-1}$ can updated directly in $O(n^2)$ which yields the recursive parameter updates of FPKF shown in Algorithm 9 (adapted from the work of Scherrer and Geist, 2011 and Geist and Scherrer, 2013).

The step-size $\alpha_t = 1/t$ of Kalman filtering basically assumes a stationary regression problem, where all targets $r_i + \gamma \boldsymbol{\phi}_{i+1}^T \boldsymbol{\theta}_i$ are equally important. However, it is beneficial to give later targets more weight, as the parameter estimates are getting more accurate and therefore the targets $r_i + \gamma \phi_{i+1}^T \boldsymbol{\theta}_i$ are becoming more reliable. Such increased influence of

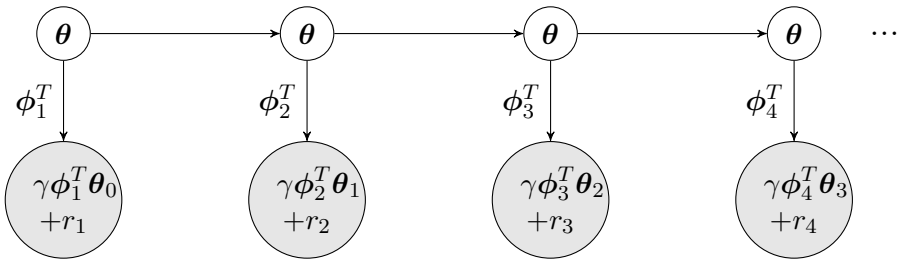


Figure 8: Illustration of the model assumptions of Fixed-Point Kalman Filtering. FPKF aims at estimating the value of the hidden variable $\boldsymbol{\theta}$ assuming a noise-free transition function (stationary environment, variable is constant over time). The main idea of FPKF is to use estimates of previous timesteps to compute the outputs $r_i + \gamma\boldsymbol{\phi}_i^T\boldsymbol{\theta}_i$ and treat them as fixed and observed in later timesteps. This assumption makes FPKF essentially different from KTD (cf. Figure 9), which only considers r_i as observed.

recent time-steps can be achieved by using a step-size α_t which decreases slower than $1/t$, for example, $a/(a+t)$ for some large a.

Bellman Residual Minimization. Bellman residual minimization (BRM) was one of the first approaches for approximate policy evaluation proposed by Schweitzer and Seidmann (1985). It calculates the least-squares solution of the MSBE given by

$$\boldsymbol{\theta} = \underbrace{(\boldsymbol{\Delta\Phi}^T\boldsymbol{D}\boldsymbol{\Delta\Phi})}_{\boldsymbol{F}}^{-1}\underbrace{\boldsymbol{\Delta\Phi}^T\boldsymbol{D}\boldsymbol{R}}_{\boldsymbol{g}}, \tag{31}$$

where $\boldsymbol{\Delta\Phi} = \boldsymbol{\Phi} - \gamma\boldsymbol{P}^\pi\boldsymbol{\Phi}$ denotes the difference of the state features and the expected next state features discounted by γ. Again, the matrices $\boldsymbol{F}$ and $\boldsymbol{g}$ can be estimated by samples, similar to $\boldsymbol{A}$ and $\boldsymbol{b}$ of LSTD, that is,

$$\boldsymbol{F}_t = \sum_{k=0}^{t}(\boldsymbol{\phi}_k - \gamma\boldsymbol{\phi}'_{k+1})(\boldsymbol{\phi}_k - \gamma\boldsymbol{\phi}''_{k+1})^T, \quad \boldsymbol{g}_t = \sum_{k=0}^{t}(\boldsymbol{\phi}_k - \gamma\boldsymbol{\phi}'_{k+1})r''_k.$$

However, as the residual-gradient algorithm, BRM suffers from the double-sampling problem because $\boldsymbol{F}_t$ contains the product $\boldsymbol{\phi}_{k+1}\boldsymbol{\phi}_{k+1}^T$ of the features of the next state and $\boldsymbol{g}_t$ the product $\boldsymbol{\phi}_{k+1}r_k$ (see Section 2.1 and the discussion of the residual-gradient algorithm in Section 2.2.1). It therefore minimizes the MSTDE if we use one successor state sample, that is, set $\boldsymbol{\phi}'_{k+1} = \boldsymbol{\phi}''_{k+1}$. To converge to a minimum of the MSBE, we have to use two independent samples s'_{k+1}, r'_k and s''_{k+1} r''_k. For this reason, the BRM algorithm can only be employed for either a finite state space where we can visit each state multiple times or if we know the model of the MDP. See Algorithm 10 in Appendix C for the recursive update rules with double samples. For updates with a single sample see Algorithm 11, which already includes eligibility traces (from Scherrer and Geist, 2011; Geist and Scherrer, 2013, see also Section 2.4.1). If we compare the least-squares solutions for the MSPBE and the MSBE, we

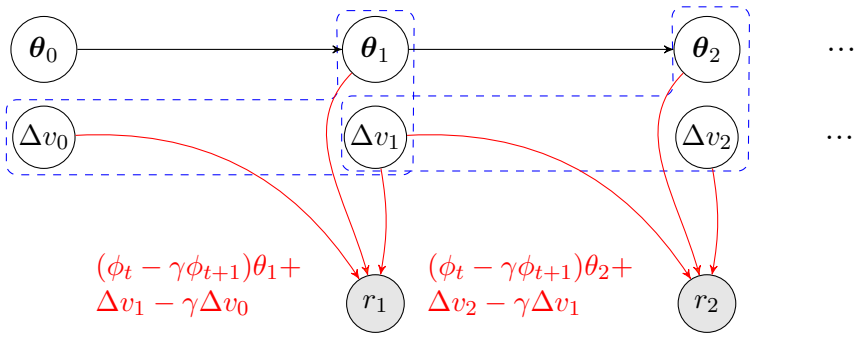


Figure 9: Graphical Model of Kalman TD learning and Gaussian-process TD learning for linearly parameterized value functions. Both approaches assume that a Gaussian process generates the random variables and linear function approximation $v_t = \boldsymbol{\theta}_t^T \phi_t$ is used. KTD aims to track the hidden state (blue dashed set of variables) at each time step given the reward observation generated by the relationship (bend arrows, in red) of the Bellman equation. While GPTD assumed $\boldsymbol{\theta}$ to be constant over time, that is, $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t$, KTD allows changing parameters.

can see that the product of $\mathbf{\Delta\Phi}^T\mathbf{\Delta\Phi}$ cancels out for the MSPBE due to the projection of the MSBE in the feature space and the MSPBE can subsequently avoid the double-sampling problem.

2.2.3 Probabilistic Models

While gradient-based and least-squares approaches are motivated directly from an optimization point of view, probabilistic methods take a different route. They build a probabilistic model and infer value function parameters which are most likely, given the observations. These methods not only yield parameter estimates that optimize a cost function, but also provide a measure of uncertainty of these estimates. Especially in policy iteration, this information can be very helpful to decide whether more observations are necessary or the value function estimate is sufficiently reliable to improve the policy.

Gaussian-process temporal-difference learning (GPTD) by Engel et al. (2003, 2005) assumes that the rewards r_t as well as the unknown true values of the observed states $v_t = V(s_t)$ are random variables generated by a Gaussian process. The Bellman Equation (2) specifies the relation between the variables

$$\tilde{v}_t := v_t + \Delta v_t = r_t + \gamma(v_{t+1} + \Delta v_{t+1}), \tag{32}$$

where $\tilde{v}_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_t$ is the future discounted reward of the current state s_t in the particular trajectory. The difference between $\tilde{v}_t$ and the average future discounted reward v_t is denoted by Δv_t and originates from the uncertainty in the system, that is, the policy π and the state transition dynamics $\mathcal{P}$. Please see Figure 9 for a graphical model illustrating the dependencies of the random variables. While $\Delta v_t \sim \mathcal{N}(0, \sigma^2)$ always has mean zero, we have to set its variances σ_t a-priori based on our belief of π and $\mathcal{P}$. The covariance $\mathbf{\Sigma}_t$ of all

Δv_i for $i = 1, \dots, t$ is a band matrix with bandwidth 2 as the noise terms of two subsequent time steps are correlated.

We consider again a linear approximation of the value function, that is, $v_t = \boldsymbol{\phi}_t^T \boldsymbol{\theta}$. Prior knowledge about the parameter $\boldsymbol{\theta}$ can be incorporated in the prior $p(\boldsymbol{\theta})$, which acts as a regularizer and is usually set to $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. GPTD infers the mean and covariance of the Gaussian distribution of $\boldsymbol{\theta}$ given the observations $r_0, \dots, r_t$ and our beliefs. The mean value corresponds to the maximum a-posteriori prediction and is equivalent to finding the solution of the regularized linear regression problem

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}} \|\Delta \boldsymbol{\Phi}_t \boldsymbol{\theta} - \boldsymbol{r}_t\|^2_{\boldsymbol{\Sigma}_t^{-1}} + \|\boldsymbol{\theta}\|^2, \tag{33}$$

where $\boldsymbol{r}_t = [r_0, r_1, \dots, r_t]^T$ is a vector containing all observed rewards. The matrix $\Delta \boldsymbol{\Phi}_t = [\Delta \boldsymbol{\phi}_0, \dots, \Delta \boldsymbol{\phi}_t]^T$ is the difference of features of all transitions with $\Delta \boldsymbol{\phi}_t = \boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}$. The solution of this problem can be formulated as

$$\boldsymbol{\theta}_t = \left(\Delta \boldsymbol{\Phi}_t \boldsymbol{\Sigma}_t^{-1} \Delta \boldsymbol{\Phi}_t^T + \boldsymbol{I}\right)^{-1} \Delta \boldsymbol{\Phi}_t \boldsymbol{\Sigma}_t^{-1} \boldsymbol{r}_t. \tag{34}$$

At first glance, the solution is similar to the MSBE least-squares solution. However, the noise terms are often highly correlated and, hence, $\boldsymbol{\Sigma}_t^{-1}$ is not a diagonal matrix. We can transform the regression problem in Equation (33) into a standard regularized least-squares problem with i.i.d. sampled data points by a whitening transformation (for details see Appendix B). We then see that the whitening transforms the reward vector $\boldsymbol{r}_t$ into the vector of the long term returns $\boldsymbol{R}_t$ where the hth elements corresponds to $(\boldsymbol{R}_t)_h = \sum_{k=h}^{t} \gamma^{k-h} r_k$. Consequently, the mean prediction of GPTD is equivalent to regularized Monte-Carlo value-function estimation (cf. Section 1.2) and minimizes the MSE from Equation (4) in the limit of infinite observations. For finite amount of data, the prior on $\boldsymbol{\theta}$ compensates the high variance problem of Monte-Carlo estimation, but may also slow down the learning process.

The quantity in Equation (34) can be computed incrementally without storing $\Delta \boldsymbol{\Phi}_t$ explicitly or inverting a $n \times n$-matrix at every timestep by a recursive algorithm shown in Algorithm 12. Its full derivation can be found in Appendix 2.1 of Engel (2005). The most expensive step involves a matrix product of the covariance matrix $\boldsymbol{P}_t \in \mathbb{R}^{n \times n}$ of $\boldsymbol{\theta}_t$ and $\Delta \boldsymbol{\phi}_{t+1}$. Thus, GPTD has a runtime complexity of $O(n^2)$.

Kalman temporal-difference learning (KTD) by Geist and Pietquin (2010) is another probabilistic model very similar to GPTD. While it is based on the same assumptions of Gaussian distributed random variables, it approaches the value estimation problem from a filtering or signal processing perspective. KTD uses a Kalman Filter to track a hidden state, which changes over time and generates the observed rewards at every timestep. The state consists of the parameter to estimate $\boldsymbol{\theta}_t$ and the value difference variables $\Delta v_t, \Delta v_{t-1}$ with $\Delta v_t = \tilde{v}_t - \boldsymbol{\phi}_t \boldsymbol{\theta}_t$ of the current and last timestep. As in GPTD, the rewards are generated from this state with Equation (32) resulting in the linear observation function g

$$r_t = g(\boldsymbol{\theta}_t, \Delta v_t, \Delta v_{t+1}) = [\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}] \boldsymbol{\theta}_t + \Delta v_t - \gamma \Delta v_{t+1}.$$

KTD does not necessarily assume that a unique single parameter $\boldsymbol{\theta}$ has created all rewards, but allows the parameter to change over time (as if the environment is non-stationary). More precisely, $\boldsymbol{\theta}_{t+1} \sim \mathcal{N}(\boldsymbol{\theta}_t, \boldsymbol{\Sigma}_\theta)$ is modeled as a random walk. As we focus on stationary environments, we can set $\boldsymbol{\Sigma}_\theta = \mathbf{0}$ to enforce constant parameters and faster convergence.

In this case, KTD and GPTD are identical algorithms for linear value function parametrization (cf. Algorithm 12). The graphical model in Figure 9 illustrates the similar assumptions of both approaches. Besides KTD's ability to deal with non-stationary environments, KTD and GPTD differ mostly in the way they handle value functions that are non-linear in the feature space. KTD relies on the unscented transform (a deterministic sample approximation for nonlinear observation functions), while GPTD avoids explicit function parametrization assumptions with kernels (cf. Section 2.3). Depending on the specific application and available domain knowledge, either a well-working kernel or a specific nonlinear parametrization is easier to chose.

Both probabilistic approaches share the benefit of not only providing a parameter estimate but also an uncertainty measure on it. However, as they optimize the mean squared error similar to Monte Carlo value-function estimation, their estimates may suffer from higher variance. The long-term memory effect originating from the consideration of all future rewards also prevents off-policy learning as discussed by Geist and Pietquin (2010, Section 4.3.2) and Engel (2005).

2.3 Feature Handling

The feature representation $\boldsymbol{\phi}$ of the states has a tremendous influence, not only on the quality of the final value estimate but also on convergence speed. We aim for features that can represent the true value function accurately and are as concise as possible to reduce computational costs and the effects of over-fitting. Many commonly used feature functions are only locally active. Their components are basis functions which have high values in a specific region of the state space and low ones elsewhere. For example, cerebellar model articulation controllers (CMAC) cover the state space with multiple overlapping tilings (Albus, 1975), also known as tile-coding. The feature function consists of binary indicator functions for each tile. Alternatively, smoother value functions can be obtained with radial basis functions. Such bases work well in practice, but often only if they are normalized such that $\|\phi(s)\|_1 = 1$ for all states $s \in \mathcal{S}$ as discussed by Kretchmar and Anderson (1997). The performance of many algorithms, including the regularization methods discussed in Section 2.3.2, can be improved by using normalized features with zero mean and unit variance.

Local function approximators are limited to small-scale settings as they suffer from the curse of dimensionality similar to exact state representations (cf. Section 1.2). When the number of state dimensions increases, the number of features explodes exponentially and so does the amount of data required to learn the value function. Therefore, recent work has focused on facilitating the search for well-working feature functions. These efforts follow two principled approaches: (1) features are either generated automatically from the observed data or (2) the learning algorithms are adapted to cope with huge numbers of features efficiently in terms of data and computation time. We briefly review the advances in both directions in the following two sections.

2.3.1 Automatic Feature Generation

Kernel-based value function estimators represent the value of a state in terms of the similarity of that state to previously observed ones, that is, at each time step the similarity to current state is added as an additional feature. A well chosen kernel, that is, the distance or similarity

measure, is crucial for the performance of kernel-based approaches, as well as an adequate sparsification technique to prevent the number of features to grow unboundedly.

GPTD (Engel et al., 2003) and LSTD (Xu et al., 2005, known as Kernelized LSTD, KLSTD) have been extended to use kernelized value functions. A similar approach was proposed in the work of Rasmussen and Kuss (2003) where a kernel-based Gaussian process is used for approximating value functions based on the Bellman Equation (2). This approach, KLSTD and GPTD were unified in a model-based framework for kernelized value function approximation by Taylor and Parr (2009). Jung and Polani (2006) introduced an alternative online algorithm originating from least-squares support-vector machines to obtain the GPTD value function estimate; however, it is limited to MDPs with deterministic transitions.

An alternative to kernel methods based on spectral learning was presented by Mahadevan and Maggioni (2007). The authors proposed to build a graph-representation of the MDP from the observations and chose features based on the eigenvector of the Graph-Laplacian. Compared to location-based features such as radial basis functions, this graph-based technique can handle discontinuities in the value-function more accurately. In contrast, Menache et al. (2005) assumes a fixed class of features, for example, RBFs, and optimizes only the free parameters (e.g., the basis function widths) by gradient descent or by using the cross-entropy optimization method (De Boer et al., 2010). Keller et al. (2006) uses neighborhood component analysis, a dimensionality reduction techniques for labeled data, to project the high-dimensional state space to a lower dimensional feature representation. They take the observed Bellman errors from Equation (7) as labels to obtain features that are most expressive for the value function. The approaches of Parr et al. (2007), Painter-Wakefield and Parr (2012a) and Geramifard et al. (2013, 2011) are based on the orthogonal matching principle (Pati et al., 1993) and incrementally add features which have high correlation with the temporal-difference error. The intuition is that those additional features enable the algorithms to further reduce the temporal-difference error.

2.3.2 Feature Selection by Regularization

Value function estimators face several challenges when the feature space is high dimensional. First, the computational costs may become unacceptably large. Second, a large number of noise-like features deteriorates the estimation quality due to numerical instabilities and, finally, the amount of samples required for a reliable estimate grows prohibitively. The issues are particularly severe for least-squares approaches which are computationally more involved and tend to over-fit when the number of observed transitions is lower than the dimensionality of the features.

The problem of computational costs for second order methods can be addressed by calculating the second order updates incrementally. For example, the parameter update of incremental LSTD (iLSTD proposed by Geramifard et al., 2006a,b) is linear in the total number of features ($O(n)$ instead of $O(n^2)$ for standard LSTD) if only a very small number of features is non-zero in each state. Most location based features such as CMAC or fixed-horizon radial basis functions fulfill this condition.

Information theoretic approaches which compress extensive feature representations are prominent tools in machine learning for reducing the dimensionality of a problem. Yet, these methods are often computationally very demanding which limits their use in online

	Formulation		Optimization Technique
LSTD with ℓ_2	$f(\boldsymbol{\theta}) \propto \|\boldsymbol{\theta}\|_2^2$	$g(\boldsymbol{\omega}) = 0$	closed form solution (Bradtke and Barto, 1996)
LSTD with ℓ_2,ℓ_2	$f(\boldsymbol{\theta}) \propto \|\boldsymbol{\theta}\|_2^2$	$g(\boldsymbol{\omega}) \propto \|\boldsymbol{\omega}\|_2^2$	closed form solution (Hoffman et al., 2011)
LARS-TD	$f(\boldsymbol{\theta}) \propto \|\boldsymbol{\theta}\|_1$	$g(\boldsymbol{\omega}) = 0$	custom LARS–like solver (Kolter and Ng, 2009)
LC-TD	$f(\boldsymbol{\theta}) \propto \|\boldsymbol{\theta}\|_1$	$g(\boldsymbol{\omega}) = 0$	standard LCP solvers (Johns et al., 2010)
ℓ_1-PBR	$f(\boldsymbol{\theta}) = 0$ (*)	$g(\boldsymbol{\omega}) \propto \|\boldsymbol{\omega}\|_1$	standard Lasso solvers (Geist and Scherrer, 2011)
LSTD with ℓ_2,ℓ_1	$f(\boldsymbol{\theta}) \propto \|\boldsymbol{\theta}\|_2^2$	$g(\boldsymbol{\omega}) \propto \|\boldsymbol{\omega}\|_1$	standard Lasso solvers (Hoffman et al., 2011)
Laplacian-based reg. LSTD	$f(\boldsymbol{\theta}) \propto \|\boldsymbol{L}\boldsymbol{\Phi}_t\boldsymbol{\theta}\|_2^2$	$g(\boldsymbol{\omega}) = 0$	closed form solution (Geist et al., 2012)
LSTD-ℓ_1	$\min t\|\boldsymbol{A}\boldsymbol{\theta} - \boldsymbol{b}\|_2^2 + \mu\|\boldsymbol{\theta}\|_1$		standard Lasso solvers (Pires, 2011)
D-LSTD	$\min \|\boldsymbol{\theta}\|_1$ s.t. $t\|\boldsymbol{A}\boldsymbol{\theta} - \boldsymbol{b}\|_\infty \leq \mu$		standard LP solvers (Geist et al., 2012)

Table 2: Comparison of Regularization Schemes for LSTD. f and g are the regularization terms in the nested problem formulation of LSTD (Equations 2.3.2 and 2.3.2). Parameters μ control the regularization strength. (*) ℓ_1-PBR actually assumes a small ℓ_2 regularization on the operator problem if the estimate of $\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Phi}$ is singular, which is usually the case for $t < m$.

reinforcement learning. Some information theoretic approaches are equivalent to a special form of regularization. Regularization is a standard way to avoid over-fitting by adding punishment terms for large parameters $\boldsymbol{\theta}$. The regularization point of view often leads to computationally cheaper algorithms compared to information theory. Hence, there has been increasing interest in adding different regularization terms to LSTD and similar algorithms (cf. Table 2). As in supervised learning, the most common types of regularization terms are ℓ_1 and ℓ_2-regularization, which penalize large ℓ_1 respective ℓ_2 norms of the parameter vector. While ℓ_2-regularization still allows closed form solutions, it becomes problematic when there are only very few informative features and a high number of noise-like features. Regularizing with ℓ_2-terms usually yields solutions with small but non-zero parameters in each dimension, which have low quality when there are many noise-like features. ℓ_1-regularization on the other hand prevents closed form solutions, but is known to induce sparsity for the resulting estimate of $\boldsymbol{\theta}$ where only few entries are different from zero. Hence, ℓ_1-regularization implic-

itly performs a feature selection and can cope well with many irrelevant features. Therefore, it is well suited for cases where the number of features exceeds the number of samples.

Most regularization methods are derived from the nested OPE-FPE optimization formulation of LSTD in Equations (13)–(14) where the regularization term is added either to the FPE problem, to the OPE problem or in both problems[3]

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{V}_{\boldsymbol{\theta}} - T\boldsymbol{V}_{\hat{\boldsymbol{\omega}}}\|_{\boldsymbol{D}}^2 + \frac{1}{t} f(\boldsymbol{\theta}) \quad \text{and}$$
$$\hat{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega}} \|\boldsymbol{\Phi}(\hat{\boldsymbol{\theta}} - \boldsymbol{\omega})\|_{\boldsymbol{D}}^2 + \frac{1}{t} g(\boldsymbol{\omega}).$$

Using the regularization term $f(\boldsymbol{\theta})$ corresponds to regularization before setting the fixpoint solution in the OPE problem while enabling $g(\boldsymbol{\omega})$ regularizes after employing the fixpoint solution. Using an ℓ_2-penalty in $f(\boldsymbol{\theta})$, that is, $f(\boldsymbol{\theta}) = \beta_f \|\boldsymbol{\theta}\|_2^2$ yields the solution $\hat{\boldsymbol{\theta}} = (\boldsymbol{A} + \beta_f t^{-1} \boldsymbol{I})^{-1} \boldsymbol{b}$. This form of regularization is often considered as the standard regularization approach for LSTD, since it is equivalent to initializing $\boldsymbol{M}_0 = \boldsymbol{A}_0^{-1} = \beta_f^{-1} \boldsymbol{I}$ in the recursive LSTD algorithm (Algorithm 5). The posterior mean of GPTD also corresponds to LSTD with an ℓ_2-regularization of the operator problem if both algorithms are extended with eligibility traces (see the next section). In addition to the regularization of the operator problem, Farahmand et al. (2008) and Hoffman et al. (2011) proposed an ℓ_2-penalty for the fixpoint problem (i.e., $g(\boldsymbol{\omega}) = \beta_g \|\boldsymbol{\omega}\|_2^2$). There are still closed-form solutions for both problems with ℓ_2-regularizations. However, the benefits of such regularization in comparison to just using $f(\boldsymbol{\theta})$ still need to be explored.

Regularization with ℓ_1-norm was first used by Kolter and Ng (2009) in the operator problem, that is, $f(\boldsymbol{\theta}) = \beta_f \|\boldsymbol{\theta}\|_1$ and $g(\boldsymbol{\omega}) = 0$. They showed that ℓ_1-regularization gives consistently better results than ℓ_2 in a policy iteration framework and is computationally faster for a large number of irrelevant features. Yet, using ℓ_1-regularization for the OPE problem prevents a closed form solution and the resulting optimization problem called Lasso-TD is non-convex. The least-angle regression algorithm (Efron et al., 2004) could be adapted to solve this optimization problem which yielded the LARS-TD algorithm (Kolter and Ng, 2009). Johns et al. (2010) started from the Lasso-TD problem but reformulated it as a linear complementarity problem (Cottle et al., 1992), for which standard solvers can be employed. Additionally, this linear complementary TD (LC-TD) formulation allows using warm-starts when the policy changes.[4] Ghavamzadeh et al. (2011) showed that the Lasso-TD problem has a unique fixpoint which means that LC-TD and LARS-TD converge to the same solution. In addition, Ghavamzadeh et al. (2011) provided bounds on the MSE for this fixpoint.

The ℓ_1-Projected-Bellman-Residual (ℓ_1-PBR) method (Geist and Scherrer, 2011) puts the regularization onto the fixpoint problem instead of the operator problem, that is, $f(\boldsymbol{\theta}) = 0$ and $g(\boldsymbol{\omega}) = \beta_g \|\boldsymbol{\omega}\|_1^2$. Hoffman et al. (2011) proposed a similar technique but with additional ℓ_2-penalty on the operator problem. Regularizing FPE problem with an ℓ_1 norm allows for a closed form solution of the OPE problem. Using this solution in the regularized FPE problem reduces to a standard Lasso problem and, hence, a standard Lasso solver

3. For notational simplicity, we slightly abuse notation and use the true OPE and FPE objectives instead of the sample-approximations at time t.
4. Warm-starts are valuable in policy iteration: The solution of the last policy can be used to substantially speed-up the computation of the value function for the current policy.

can be employed instead of specialized solution as for the Lasso-TD problem. Furthermore, Lasso-TD has additional requirements on the $\boldsymbol{A}$-matrix of LSTD[5] which generally only hold in on-policy learning. Approaches with ℓ_1-regularized operator problems do not have this limitation and only make mild assumptions in off-policy settings. Despite these theoretical benefits, empirical results indicate comparable performance to the Lasso-TD formulation and, hence, ℓ_1-regularization for the FPE problem is a promising alternative.

Petrik et al. (2010) propose using ℓ_1-regularization in the linear program formulation of dynamic programming for finding the value function. However, their analysis concentrates on the case where the transition kernel $\mathcal{P}^\pi$ is known or approximated by multiple samples. Another family of methods considers the linear system formulation of LSTD $\boldsymbol{A\theta} = \boldsymbol{b}$ directly. Pires (2011) suggests to solve this system approximately with additional ℓ_1-regularization

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{A\theta} - \boldsymbol{b}\|_2^2 + \frac{\beta}{t}\|\boldsymbol{\theta}\|_1.$$

Again, this problem is a standard convex Lasso problem solvable by standard algorithms and applicable to off-policy learning. Dantzig-LSTD (D-LSTD, Geist et al., 2012) takes a similar approach and considers

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{\theta}\|_1 \quad \text{subject to} \quad \|\boldsymbol{A\theta} - \boldsymbol{b}\|_\infty \leq \frac{\beta}{t}.$$

This optimization problem, a standard linear program, is motivated by the Dantzig selector of Candes and Tao (2005) and can be solved efficiently. It is also well-defined for off-policy learning. The aim of this problem is to minimize the sum of all parameters while making sure that the linear system of LSTD is not violated by more than βt^{-1} in each dimension.

All regularization approaches so far have treated each parameter dimension equally. However, it might be helpful to give the parameter components different weights. Johns and Mahadevan (2009) suggested to use the Laplacian $\boldsymbol{L}$ of the graph-based representation of the MDP as weights and add $\beta_L\|\boldsymbol{L\Phi\theta}\|_{\boldsymbol{D}}^2$ as an additional term to the MSPBE. Investigating the benefits of other problem-dependent weighted norms are left for future work. The performance of all regularization schemes strongly depends on the regularization strength β, which has to be specified by hand, found by cross-validation or set with the method of Farahmand and Szepesvári (2011).

Instead of regularizing the value-function estimation problem, we could also estimate the value function directly with LSTD in a lower-dimensional feature space. Ghavamzadeh et al. (2010) showed in a theoretical analysis that projecting the original high-dimensional features to a low-dimensional space with a random linear transformation (LSTD with Random Projections, LSTD-RP) has the same effect as regularization. However, no empirical results for this algorithm are given. Alternatively, features can be selected explicitly to form the lower-dimensional space. Hachiya and Sugiyama (2010) proposed to consider the conditional mutual information between the rewards and the features of observed states and provided an efficient approximation scheme to select a good subset as features.

There have also been efforts to regularize Bellman residual minimization. Loth et al. (2007) added an ℓ_1-penalty to the MSBE and proposed a gradient-based technique to find

5. The matrix has to be a P-matrix. P-matrices, a generalization of positive definite matrices, are square matrices with all of their principal minors positive.

the minimum incrementally. In contrast, Farahmand et al. (2008) regularized with an ℓ_2-term to obtain the optimum in closed form. Gradient-based TD-algorithms are less prone to over-fitting than least-squares approaches when few transitions are observed as the norm of the parameter vector is always limited for a small number of updates. However, if the observed transitions are re-used by running several sweeps of stochastic gradient updates, regularization becomes as relevant as for the least-squares approaches. In addition, if a large number of features are irrelevant for the state value, the gradient and especially its stochastic approximation becomes less reliable. Therefore, there has been recent interest in promoting sparseness by adding a soft-threshold shrinkage operator to gradient-based algorithms (Painter-Wakefield and Parr, 2012b; Meyer et al., 2012) and reformulating the regularized objective as a convex-concave saddle-point problem (Liu et al., 2012).

Despite the extensive work on regularization schemes for LSTD, many directions still need to be explored. For example, many feature spaces in practice have an inherent structure. They may for instance consist of multiple coverings of the input space with radial basis functions of different widths. There has been work on exploiting such structures with hierarchical regularization schemes in regression and classification problems (Zhao et al., 2009; Jenatton et al., 2010). These approaches divide the parameters into groups and order the groups in a hierarchical structure (e.g., trees), which determines the regularization in each group. While such schemes have been successfully applied to images and text documents, it is an open question whether they can be adapted to work online and to which extent policy evaluation tasks could benefit from hierarchical regularization.

2.4 Important Extensions

In the previous sections, temporal-difference methods have been introduced in their most basic form to reveal the underlying ideas and avoid cluttered notation. We now introduce two extensions which are applicable to most methods. First, we briefly discuss eligibility traces for improving the learning speed by considering the temporal difference of more than one timestep. Subsequently, importance-reweighting is presented which enables temporal-difference methods to estimate the value function from off-policy samples. While the aim of this paper is giving a survey of existing methods, we will also present an alternative implementation of importance reweighting for LSTD and TDC which considerably decreases the variance of their estimates. We focus on the purpose and the functionality of eligibility traces and importance reweighting and, hence, illustrate their actual implementation only for selected TD methods.

2.4.1 Eligibility Traces

Eligibility traces (e-traces, Sutton, 1988) are an efficient implementation of blending between TD methods and Monte-Carlo sampling. To understand the purpose of this blending, it is beneficial to first identify the different sources of errors in TD methods. While we derived the update rules of the algorithms based on observed samples directly from the underlying cost functions such as MSE, MSBE or MSPBE, we now make the sample-based approximations of the objective functions explicit. These approximations at a given timestep t are denoted by a subscript t, for example, MSE_t, MSBE_t or MSPBE_t (see also Figure 5).

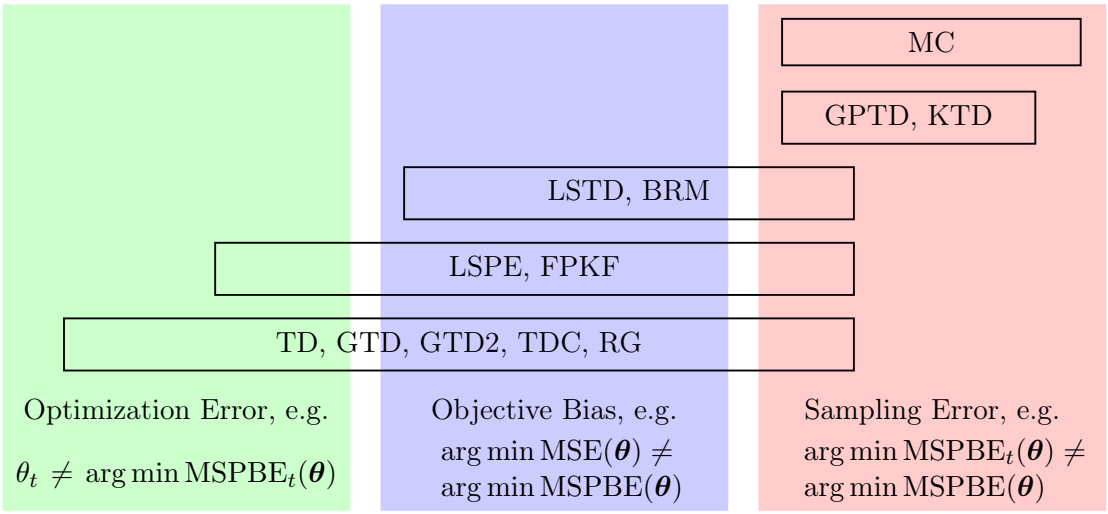


Figure 10: Visualization of Conceptual Error Sources for Policy Evaluation Methods: The *sampling error* is always present and accounts for the approximation of the chosen objective function with observed samples. The higher the variance of these samples the higher the sampling error. If the objective of the method is not directly the MSE, the method will suffer from the *objective bias*. The *optimization error* is present for methods which do not find the minimum of the approximated objective function directly, for example, gradient-based approaches. The positions of the boxes and their overlap with the shaded areas denote the extent, to which the respective methods suffer from each error source. Note that the actual amount of each error source is not visualized and varies drastically between different MDPs, feature representations, policies and number of time steps. MSPBE_t denotes the approximation of the MSPBE with samples observed at timesteps 1 to t.

Error Decomposition. Leaving numerical issues aside, there are three conceptual sources of errors as illustrated in Figure 10. Consider for example Monte-Carlo sampling, which does not rely on temporal differences, but simply takes the observed accumulated reward as a sample for each state. Hence, at time t, it finds the parameter estimate by computing the minimum of

$$\mathrm{MSE}_t(\boldsymbol{\theta}) = \sum_{i=0}^{t} \left(\boldsymbol{\phi}_i^T \boldsymbol{\theta} - \sum_{k=i}^{t} \gamma^{k-i} r_k \right)^2 .$$

After infinitely many time steps, this sample-based approximation of the MSE converges to the true error prescribed by Equation (4), which can also be written as

$$\mathrm{MSE}(\boldsymbol{\theta}) = \left\| \boldsymbol{\Phi}\boldsymbol{\theta} - \sum_{k=0}^{\infty} \gamma^k \boldsymbol{P}^k \boldsymbol{R} \right\|_{\boldsymbol{D}}^2 . \tag{35}$$

The difference between the approximation and the true objective function, referred to as *sampling error*, is present for all methods. TD methods avoid estimating $\gamma^k \boldsymbol{P}^k \boldsymbol{R}$ for $k > 0$ directly by replacing these terms with the value function estimate, that is, use bootstrapping with the Bellman operator T. As the replaced terms cause the high variance, the sampling error decreases at the price of a possible increase in the *objective bias*. This bias denotes the difference between the minimum of the TD objective function (such as MSPBE or MSBE) and the true minimum of the MSE. The regularization with priors in GPTD and KTD is an alternative for reducing the variance at the price of a temporary objective bias. Descent approaches such as the gradient methods, LSPE or FPKF do not compute the minimum of the current objective approximation analytically, but only make a step in its direction. Hence, they suffer from an additional *optimization error*. Although the errors caused by each source do not add up, but may counterbalance each other, it is a reasonable goal to try to minimize the effect of each source.

The magnitude of each type of error depends on the actual MDP, feature representation, policy and number of observed transitions. For example, the objective bias of MSPBE or MSBE vanishes for features that allow representing the true value function exactly. On the other hand, a setup where the MDP and policy are deterministic has zero variance for $\gamma^k \boldsymbol{P}^k \boldsymbol{R}$ and hence, introducing a bias with bootstrapping does not pay off. By interpolating between TD methods and Monte-Carlo estimates, we can often find an algorithm where the effects of sampling error and objective bias is minimized. The natural way to do so is to replace $\gamma^k \boldsymbol{P}^k \boldsymbol{R}$ only for terms $k > h$ in Equation (35), which yields the *h-step Bellman operator*

$$T^h \boldsymbol{V} = \underbrace{TT \dots T}_{h \text{ times}} \boldsymbol{V} = \gamma^h \boldsymbol{P}^h \boldsymbol{V} + \sum_{k=0}^{h-1} \gamma^k \boldsymbol{P}^k \boldsymbol{R}.$$

As it considers the h future rewards, it is also referred to as h-step look-ahead (Sutton and Barto, 1998). If we used the h-step Bellman operator to redefine the objective functions (e.g., MSBE or MSPBE), we would need to observe the rewards $r_t, r_{t+1}, \dots r_{t+h-1}$ and state s_{t+h} before we could use s_t for estimation, that is, approximating $T^h V(s_t)$ with a sample corresponds to

$$T^h V(s_t) \approx \gamma^h V(s_{t+h}) + \sum_{k=0}^{h-1} \gamma^k r_{t+k}.$$

Hence, online estimation is not possible for large h. Eligibility traces circumvent this problem and allow taking each sample into account immediately.

Eligibility Traces. Eligibility traces rely on the λ-*Bellman operator* T_λ defined as a weighted average over all T^k

$$T_\lambda = (1 - \lambda) \sum_{k=0}^{\infty} \lambda^k T^{k+1}. \tag{36}$$

The term $(1-\lambda)$ is a normalization factor which ensures that all weights sum to 1. TD methods extended with eligibility traces minimize objectives redefined on this average Bellman

operator such as the MSPBE$^\lambda$ objective

$$\text{MSPBE}^\lambda(\boldsymbol{\theta}) = \|\boldsymbol{V}_{\boldsymbol{\theta}} - \Pi T_\lambda \boldsymbol{V}_{\boldsymbol{\theta}}\|^2_{\boldsymbol{D}}.$$

For $\lambda = 0$ only $T^1 = T$ is used and, hence, MSPBE^0 corresponds to the standard MSPBE. Only considering T^∞ in the objective corresponds to the MSE from Equation (35). Due to the discount factor γ, there exists a K such that $\|T^k \boldsymbol{V}\|$ deviates less than a small constant from $\|T^\infty \boldsymbol{V}\|$ for all $k > K$. For $\lambda = 1$, the terms $k > K$ in Equation (36) are given infinitely more weight than $k \leq K$ and hence $\lim_{\lambda \to 1} T_\lambda = T^\infty$. We realize that the MSPBE^1 is equivalent to the MSE.

The basic idea of eligibility traces is to approximate the k-step Bellman operators in the weighted sum with samples as soon as possible. Thus, at timestep t, state s_t is used for T^1, state s_{t-1} for T^2, s_{t-2} for T^3 and so on. The special choice of exponentially decreasing weights allows storing the previously observed states efficiently as a summed vector, a so-called eligibility trace.

Implementation for TD Learning. To illustrate the efficient approximation of T_λ and eligibility traces as compact storage of previously observed features, we consider the extension of the standard TD learning algorithm for multi-step temporal differences. Its extended update rule is given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \delta_t \sum_{k=0}^{t} (\lambda\gamma)^k \boldsymbol{\phi}_{t-k}. \tag{37}$$

The parameter λ puts more weight on more recent states. As shown by Sutton and Barto (1998), the multi-step look-ahead (forward view), that is, considering future rewards in the Bellman operator, can also be understood as propagating the temporal-difference error backwards in time (often called the *backward view*), that is, updating the value of states observed before. The update in Equation (37) can be implemented efficiently by computing the sum $\sum_{k=0}^{t} (\lambda\gamma)^k \boldsymbol{\phi}_{t-k}$ incrementally. More precisely, the eligibility trace vector $\boldsymbol{z}_t$ stores the past activations of the features and is updated by

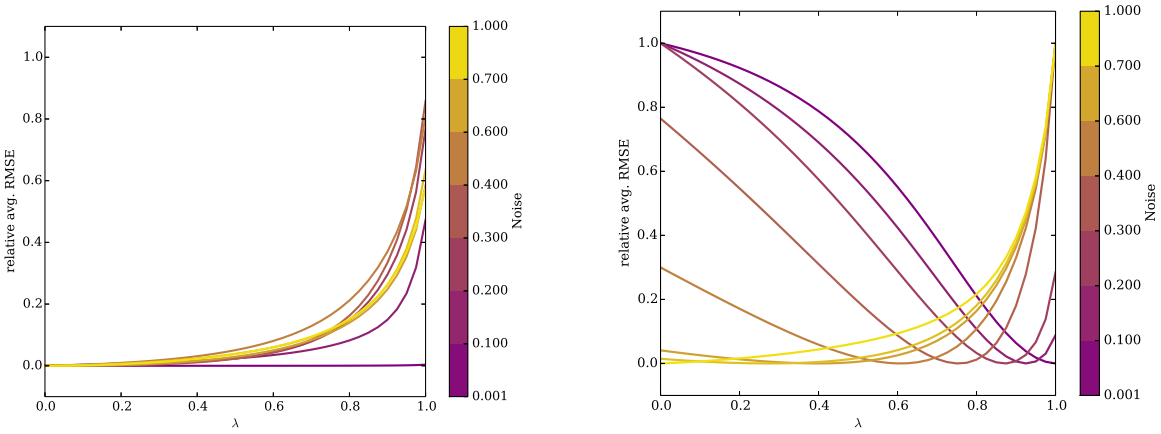
$$\boldsymbol{z}_{t+1} = \boldsymbol{\phi}_t + \lambda\gamma \boldsymbol{z}_t.$$

Updating the eligibility trace in such a way ensures that $\boldsymbol{z}_{t+1} = \sum_{k=0}^{t} (\lambda\gamma)^k \boldsymbol{\phi}_{t-k}$ for all timesteps. The update rule of TD learning can then be written more concisely with eligibility traces as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \delta_t \boldsymbol{z}_{t+1}.$$

The TD learning algorithm with a certain setting of λ is often referred to as TD(λ) where TD(0) corresponds to the standard TD learning algorithm without eligibility traces.

For $\lambda > 0$, the algorithm also updates the value function at states s_h with $h < t$, which is reasonable, as the value at state s_{t-1} is very likely to change when $V(s_t)$ changes. In contrast, TD(0) learning does not reuse its data-points and would need to observe state s_{t-1} again to update the value function at state s_{t-1}. Subsequently, s_{t-2} needs to be visited again to update $V(s_{t-2})$, and so on. Hence, eligibility traces not only allow one to find the best trade-off between objective bias and sampling error, but also reduce the optimization error for gradient based approaches.



(a) Perfect feature representation. If we can approximate the value function perfectly, the bias between MSPBE and MSE is zero, and hence LSTD(0) should always be preferred.

(b) Impoverished features. Here, we have to choose a trade-off between minimizing the variance of the objective function with LSTD(0) and minimizing the bias of the objective function with LSTD(1). The optimal trade-off depends on the amount of noise in the MDP.

Figure 11: Eligibility-Traces implement a trade-off between minimizing the MSE and the MSPBE: Consider Example 2 for the description of the experimental setting. Each graph shows the average of the root of MSE (RMSE $= \sqrt{\text{MSE}}$) of LSTD(λ)-estimates over all timesteps. All curves are normalized by subtracting the minimum and dividing by the maximum. The plots show which λ settings produce lowest errors for for varying amount of stochasticity in the system, where we compare perfect and impoverished features.

Eligibility Traces for Other Algorithms. Most algorithms presented in this paper have been extended to use eligibility traces (see Table 1 for an overview). LSTD (Boyan, 2002), TDC (originally named GTD(λ) in Maei, 2011, but here referred to as TDC(λ) for clarity), FPKF (Scherrer and Geist, 2011; Geist and Scherrer, 2013) and BRM (Scherrer and Geist, 2011; Geist and Scherrer, 2013) have been extended to use multistep-lookahead with eligibility traces. LSPE(λ) (Nedic and Bertsekas, 2003) has been formulated with traces from the beginning. Eligibility traces have also been introduced in GPTD by Engel (2005).[6] Recently, eligibility-traces-versions of GTD2 and the residual-gradient algorithm named GTD2(λ) and gBRM(λ) (Geist and Scherrer, 2013) have been developed. All the algorithms above converge to a minimum of the MSPBE$^\lambda$ objective or respectively MSBE$^\lambda$, which is defined analogously.

Example 2 *We illustrate the benefit of interpolating between MSPBE and MSE in two experiments using e-traces and LSTD. Consider a discrete 40 state / 40 action MDP where actions of the agent deterministically determine its next state. In the first experiment, we use a perfect feature representation, that is, the value function can be estimated perfectly,*

6. In contrast to other methods, the basic version without e-traces corresponds to GPTD(1) and not GPTD(0).

while, in the second experiment, we use an incomplete feature representation by projecting the states linearly on a random 20-dimensional feature space. In both experiments, we evaluated the performance of different λ values for different policies. We varied the stochasticity of the policy, by interpolating between a greedy policy, which visits one state after another, and the uniform policy, which transitions to each state with equal probability.

In Figure 11a, we can see the results for the perfect feature representation. Here, the MSPBE$^\lambda$ does not cause any bias and its minimum coincides with the MSE solution. The plots show the relative MSE values for different λ settings for different levels of stochasticity in the system controlled by the linear blending coefficient between the greedy and uniform policy. The results confirm our intuition that using $\lambda = 0$ is always optimal for perfect features as the MSPBE minimization is unbiased. As the policy is the only source of stochasticity in the system, it behaves deterministically for the greedy policy and the performance is invariant to the choice of λ.

The picture changes for the imperfect feature representation where the minimization of the MSPBE$_\lambda$ causes a bias for the MSE (cf. Figure 11b). Setting $\lambda = 1$ performs best for the greedy policy as there is again no variance on the returns, and, hence, we avoid the bias of the MSPBE$_\lambda$ by directly minimizing the MSE. When gradually increasing the stochasticity of the policy, the optimal value of λ decreases and finally reaches the value of 0. This example illustrates that eligibility traces cause significant speed-ups of learning speed for LSTD(λ) and that the best trade-off between objective bias and sampling error highly depends on the intrinsic stochasticity of the MDP and policy. Hence, λ should be considered as an additional hyper-parameter to optimize for each setting.

2.4.2 Generalization to Off-Policy Learning by Importance Reweighting

In previous sections, we aimed at estimating state values V^π while observing the agent following policy π. However, in many applications we want to know V^π, but only have observed samples with actions chosen by a different policy. Estimating the state values of a different policy than the observed one is referred to as *off-policy value-function estimation* (cf. Section 1.2). For instance, we could be following an exploration policy while we want to know the value function of the optimal greedy policy. In a policy iteration scenario, we can employ off-policy policy evaluation to re-use data points collected with previous policies. Hence, off-policy value-function estimation is an important ingredient for efficiently learning control policies. A different application of off-policy estimation is intra-option learning (Sutton et al., 1998), where we can use samples from different options to update the value functions of the single options.

Importance Reweighting. Leveraging temporal-difference methods for off-policy estimation is based on the idea of *importance sampling* (cf. Glynn and Iglehart, 1989). Importance sampling is a well-known variance-reduction technique in statistics. It is used to approximate the expectation $\mathbb{E}_p[f(X)]$ of a function f with input $X \sim p$, when we cannot directly sample from $p(X)$ but have access to samples from another distribution $q(X)$. In this case, the expectation $\mathbb{E}_p[f(X)]$ can be approximated by

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

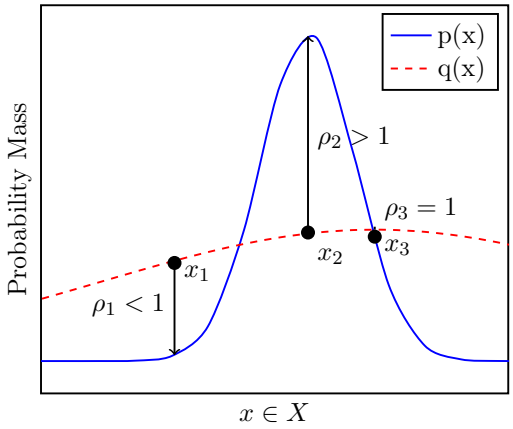


Figure 12: Importance Reweighting: Samples drawn from $q(x)$ are re-weighted by the importance weight ρ to behave like samples from $p(x)$. Data points x_1 in regions, where q is larger than p occur more frequently in the sample from q than from p and are down-weighted. In the orthogonal case x_2, the weights are larger than one and give under-represented samples more weight.

where $x_1, \dots x_M$ are realizations of q. The correctness of this statement in the limit $M \to \infty$ can be easily verified by writing out the expectations. Each sample drawn from q is re-weighted with importance weights $\rho_i = p(x_i)/q(x_i)$ to approximate $\mathbb{E}_p[f(X)]$. See Figure 12 for a visualization. The reweighting is only well-defined, if $q(X) \neq 0$ for all X with non-zero $p(X)$.

Limitations of Off-Policy Estimation. Similar to on-policy estimation, the following observation model for off-policy transitions is assumed: the departing state s_t is distributed according to the state distribution d' while the action a_t is sampled from the behavior policy π_B and the entering state s_{t+1} from the MDP dynamics $\mathcal{P}$. For on-policy value-function estimation, behavior and target policy are the same ($\pi_G = \pi_B$) and d' matches the stationary distribution of the MDP with the policy to evaluate, that is, $d' = d^{\pi_B} = d^{\pi_G}$. However, if the policies differ, the state distribution $d' = d^{\pi_B} \neq d^{\pi_G}$ is not the stationary distribution according to π_G in general. The Example from Section 1.2 in Figure 1 shows such a case.

All approaches to off-policy learning consider the difference of π_G and π_B for the actions a_t but leave the problem of the different stationary distributions unaddressed. Hence, off-policy value-function estimation does not yield the same result as on-policy estimation with samples taken from π_G, even after convergence. For example, the fixpoint of methods minimizing the MSPBE in the off-policy case can be written as

$$\mathrm{MSPBE}(\boldsymbol{\theta}) = \|\boldsymbol{V}_{\boldsymbol{\theta}} - \Pi T^{\pi_G} \boldsymbol{V}_{\boldsymbol{\theta}}\|^2_{\boldsymbol{D}_{\pi_B}}.$$

Hence, the difference of estimating the values with respect to policy π_G from off-policy or on-policy samples is the norm of the objective function. The distance metric $\boldsymbol{D}_{\pi_G}$ and $\boldsymbol{D}_{\pi_B}$ may differ substantially and therefore yield different estimates, which may be a critical

limitation of off-policy estimation. In the following, we use $d = d^{\pi_B}$ to avoid cluttered notation.

In addition, the use of importance reweighting on the policies requires $\pi_B(a|s) > 0$ for all $a \in \mathcal{A}$ and $s \in \mathcal{S}$ with $\pi_G(a|s) > 0$. Thus, each possible sample of the target policy should be observable with the behavior policy. In practice, the behavior policy is often an exploration policy and we aim for value-function estimation of a greedy policy. In this case, the restriction $\pi_B(a|s) > 0$ is not violated.

Let us now discuss specific extensions of TD learning and LSTD for off-policy learning. While we consider the algorithms without eligibility traces for notational simplicity, the derivations hold similarly for algorithms with multi-step predictions. In order to investigate the long-term behavior of algorithms, we have to consider their expected estimates, that is, their update rules in expectation of a transition (defined by the state distribution, the policy and the transition distribution).

Off-Policy TD Learning. First, consider TD learning (Algorithm 1) with the expected parameter update for on-policy learning according to π_G given by $\mathbb{E}_{\pi_G,\mathcal{P},d^{\pi_G}}[\delta_t \phi(s_t)]$. The same updates can be obtained with samples from π_B by rewriting

$$\begin{aligned}
\mathbb{E}_{\pi_G,\mathcal{P},d^{\pi_G}}[\delta_t \boldsymbol{\phi}(s_t)] &= \sum_{s_{t+1}} \sum_{a_t} \sum_{s_t} p(s_t, a_t, s_{t+1}) \delta_t \boldsymbol{\phi}(s_t) \\
&= \sum_{s_{t+1}} \sum_{a_t} \sum_{s_t} \mathcal{P}(s_{t+1}|s_t, a_t) \pi_G(a_t|s_t) d^{\pi_G}(s_t) \;\; \delta_t \boldsymbol{\phi}(s_t) \\
&\approx \sum_{s_{t+1}} \sum_{a_t} \sum_{s_t} \mathcal{P}(s_{t+1}|s_t, a_t) \pi_G(a_t|s_t) d^{\pi_B}(s_t) \;\; \delta_t \boldsymbol{\phi}(s_t) \\
&= \sum_{s_{t+1}} \sum_{a_t} \sum_{s_t} \mathcal{P}(s_{t+1}|s_t, a_t) \pi_B(a_t|s_t) d^{\pi_B}(s_t) \;\; \frac{\pi_G(a_t|s_t)}{\pi_B(a_t|s_t)} \delta_t \boldsymbol{\phi}(s_t) \\
&= \mathbb{E}_{\pi_B,\mathcal{P},d^{\pi_B}}[\rho_t \delta_t \boldsymbol{\phi}(s_t)].
\end{aligned}$$

The expectation w.r.t. the target policy π_G turned into the expectation according to the behavior policy π_B by including the importance weight $\rho_t = \pi_G(a_t|s_t)/\pi_B(a_t|s_t)$. Hence, the off-policy update rule of TD learning is given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \rho_t \delta_t \boldsymbol{\phi}(s_t).$$

Note that on-policy learning can be treated as a special case with $\pi_G = \pi_B$ and $\rho_t = 1$ for all timesteps t. As we will discuss in the following convergence analysis, TD learning might become unstable in off-policy learning. Other gradient methods have also been extended with off-policy weights and do not suffer from this drawback. The Algorithm listings in Appendix C already contain the off-policy weights for all gradient-based and least-squares algorithms.

Convergence Analysis. TD learning may be unstable, when used with a sampling distribution for the states d^{π_B} that differs from the stationary state distribution d^{π_G} induced by the Markov model to evaluate $\mathcal{P}^{\pi_G}$. Consider a batch gradient version of TD learning which

uses the expected gradient instead of the stochastic one. Results from stochastic approximation theory guarantee that the TD learning algorithm converges if batch gradient descent converges and vice versa. In addition, their fixpoints are identical. A batch-gradient step can be written as

$$\begin{aligned}\boldsymbol{\theta}_{k+1} =& \boldsymbol{\theta}_k + \alpha \mathbb{E}[\delta_t \phi_t] \\ =& \boldsymbol{\theta}_k + \alpha \mathbb{E}[(r_t + \gamma \boldsymbol{\phi}_{t+1}^T \boldsymbol{\theta}_k - \boldsymbol{\phi}_t^T \boldsymbol{\theta}_k)\boldsymbol{\phi}_t] \\ =& \boldsymbol{\theta}_k + \alpha \boldsymbol{\Phi}^T \boldsymbol{D} \left(\boldsymbol{R} + \gamma \boldsymbol{P}^{\pi_G} \boldsymbol{\Phi}\boldsymbol{\theta}_k - \boldsymbol{\Phi}\boldsymbol{\theta}_k\right) \\ =& \left(\boldsymbol{I} + \alpha \boldsymbol{A}_{\mathrm{TD}}\right) \boldsymbol{\theta}_k + \alpha \boldsymbol{b}_{\mathrm{TD}}, \end{aligned} \tag{38}$$

with $\boldsymbol{A}_{\mathrm{TD}} = \boldsymbol{\Phi}^T \boldsymbol{D} \left(\gamma \boldsymbol{P}^{\pi_G} - \boldsymbol{I}\right) \boldsymbol{\Phi}$ and $\boldsymbol{b}_{\mathrm{TD}} = \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{R}$.

The iterative update rule of Equation (38) converges if all eigenvalues of the matrix $\boldsymbol{A}_{\mathrm{TD}}$ have only negative real parts (Schoknecht, 2002). It can be shown that if $\boldsymbol{D}$ corresponds to the stationary distribution which has been generated by $\boldsymbol{P}^{\pi_G}$ this condition is satisfied, and hence, TD learning converges. However, this property of $\boldsymbol{A}_{\mathrm{TD}}$ is lost if $\boldsymbol{D}$ does not correspond to the stationary distribution, that is, $\boldsymbol{d}^{\pi_B} \neq \boldsymbol{P}^{\pi_G} \boldsymbol{d}^{\pi_B}$ and, hence, convergence can not be guaranteed for off-policy TD learning. Intuitively, TD learning does not converge because there is more weight on reducing the error of the value function for the starting states s_t of a transition than for the successor states s_{t+1}. Hence, the Bellman error in the successor states might increase, which again affects the estimation of the target values for the next parameter update. If $\boldsymbol{d} = \boldsymbol{P}^{\pi_G} \boldsymbol{d}$, the successor states have the same probability of being updated, and this problem is hence alleviated.

The second order equivalent of batch-gradient TD learning is LSPE. While both methods can be derived from the same nested optimization problem, LSPE is known to converge for off-policy policy evaluation. Hence, it is interesting to briefly look at the reason for this difference. The expected update of LSPE can be obtained from Equations (29) and (30) and is given by

$$\begin{aligned}\boldsymbol{\theta}_{k+1} =& \boldsymbol{\theta}_k + \alpha (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} \left(\boldsymbol{R} + \gamma \boldsymbol{P}^{\pi_G} \boldsymbol{\Phi}\boldsymbol{\theta}_k - \boldsymbol{\Phi}\boldsymbol{\theta}_k\right) \\ =& \left(\boldsymbol{I} + \alpha \boldsymbol{A}_{\mathrm{LSPE}}\right) \boldsymbol{\theta}_k + \alpha \boldsymbol{b}_{\mathrm{LSPE}}, \end{aligned}$$

with $\boldsymbol{A}_{\mathrm{LSPE}} = (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{A}_{\mathrm{TD}}$ and $\boldsymbol{b}_{\mathrm{LSPE}} = (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{b}_{\mathrm{TD}}$. We realize that LSPE scales the TD update with the inverse of a positive definite matrix. This scaling ensures that the matrix $\boldsymbol{A}_{\mathrm{LSPE}}$ stays negative definite and, hence, the update converges (Schoknecht, 2002). More intuitively, the second order term $(\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1}$ normalizes the TD update by the average feature activation, and, hence, overrides the problem that the successor states s_{t+1} have less probability mass than the starting states s_t of a transition.

Off-policy LSTD. We will now discuss how to adapt least-squares approaches for off-policy learning. While we show the off-policy extension for the LSTD algorithm, other methods follow analogously. LSTD relies on the estimates $\boldsymbol{A}_t$ and $\boldsymbol{b}_t$ from Equations (25) and (26) to converge to the true values $\boldsymbol{A}$ and $\boldsymbol{b}$ in Equation (24). In expectation, the estimates at

time t can be written as

$$\mathbb{E}_{d,\pi_G,\mathcal{P}}\left[\boldsymbol{A}_t\right] = \mathbb{E}_d\left[\sum_{i=0}^{t} \boldsymbol{\phi}_i \left(\boldsymbol{\phi}_i - \gamma \mathbb{E}_{\pi_G,\mathcal{P}}\left[\boldsymbol{\phi}_{i+1}\right]\right)^T\right] \quad \text{and}$$

$$\mathbb{E}_{d,\pi_G,\mathcal{P}}\left[\boldsymbol{b}_t\right] = \mathbb{E}_{d,\pi_G,\mathcal{P}}\left[\sum_{i=0}^{t} \boldsymbol{\phi}_i r_i\right].$$

We realize that the only parts which depend on the policy π_G are the terms $\mathbb{E}_{\pi_G,\mathcal{P}}\left[\boldsymbol{\phi}_{i+1}\right]$ and $\mathbb{E}_{d,\pi_G,\mathcal{P}}\left[\sum_{i=0}^{t} \boldsymbol{\phi}_i r_i\right]$. If a behavior policy π_B is used instead of π_G, these parts have to be re-weighted which results in

$$\mathbb{E}_{d,\pi_G,\mathcal{P}}\left[\boldsymbol{A}_t\right] = \mathbb{E}_d\left[\sum_{i=0}^{t} \boldsymbol{\phi}_i \left(\boldsymbol{\phi}_i - \gamma \mathbb{E}_{\pi_B,\mathcal{P}}\left[\frac{\pi_G(a_i|s_i)}{\pi_B(a_i|s_i)}\boldsymbol{\phi}_{i+1}\right]\right)^T\right] \quad \text{and}$$

$$\mathbb{E}_{d,\pi_G,\mathcal{P}}\left[\boldsymbol{b}_t\right] = \mathbb{E}_{d,\pi_B,\mathcal{P}}\left[\sum_{i=0}^{t} \frac{\pi_G(a_i|s_i)}{\pi_B(a_i|s_i)}\boldsymbol{\phi}_i r_i\right].$$

For the sample-based implementation, we arrive at the off-policy parameter estimates of LSTD proposed by Bertsekas and Yu (2009)

$$\boldsymbol{A}_t = \sum_{i=0}^{t} \boldsymbol{\phi}_i[\boldsymbol{\phi}_i - \gamma\rho_i\boldsymbol{\phi}_{i+1}]^T, \quad \boldsymbol{b}_t = \sum_{i=0}^{t} \boldsymbol{\phi}_i\rho_i r_i.$$

We refer to this off-policy reweighting as *Standard Off-Policy Reweighting*. Taking eligibility traces into account and making use of the Sherman-Morrison formula (Equation 27), the recursive off-policy version of LSTD, shown in Algorithm 5 in Appendix C, can be derived (Scherrer and Geist, 2011; Geist and Scherrer, 2013).

The $\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T$ terms in $\boldsymbol{A}_t$ are not re-weighted since it is not necessary to add importance weights to terms which do not depend on the policy for ensuring convergence to the desired solution. However, as our experiments presented in Section 3.4 show, such an approach suffers from a severe drawback. To illustrate the reason, consider the effective number of samples used to calculate the different terms. The effective number of samples for calculating the first term of $\boldsymbol{A}_t$ is always t while, for the second term, the effective number is $\varrho = \sum_i^t \rho_i$. In expectation, ϱ is equal to t and the expected estimate of $\boldsymbol{A}$ is unbiased. However, for a specific sample-based realization, ϱ will in general be different from t. As both terms in $\boldsymbol{A}_t$ are not normalized by the number of samples used for the estimate, a big part of the variance in estimating $\boldsymbol{A}_t$ will just come from the difference of ϱ to t. Despite positive theoretical analysis of the convergence properties of this reweighting strategy (Bertsekas and Yu, 2009; Yu, 2010), our experiments reveal that for more complex problems, for example, in continuous domains, the performance of LSTD with this type of reweighting breaks down due to the drastically increased variance in $\boldsymbol{A}_t$. Instead, the matrix $\boldsymbol{A}_t$ can be estimated more robustly by using the importance weight for the whole transition, that is,

$$\boldsymbol{A}_t = \sum_{i=0}^{t} \rho_i\boldsymbol{\phi}_i[\boldsymbol{\phi}_i - \gamma\boldsymbol{\phi}_{i+1}]^T, \quad \boldsymbol{b}_t = \sum_{i=0}^{t} \boldsymbol{\phi}_i\rho_i r_i.$$

A recursive method based on these updates, *LSTD Transition Off-Policy Reweighting (LSTD-TO)*, is shown in Algorithm 6. Similar reweighting strategies can be formulated for LSPE which yields *LSPE-TO* shown in Algorithm 8. To the best of our knowledge, using a transition-based reweighting for LSTD and LSPE has not been introduced in the literature so far, but is crucial for the performance of off-policy learning with least-squares methods.

3. Comparison of Temporal-Difference Methods

In this section, we compare the performance and properties of the presented policy evaluation methods quantitatively in various experiments. All algorithms were implemented in Python. The source code for each method and experiment is available at `http://github.com/chrodan/tdlearn`. In addition, further supplementary material is available at `http://www.ias.tu-darmstadt.de/Research/PolicyEvaluationSurvey`.

In Section 3.1, we present the experimental setting including the benchmark tasks and the evaluation process. Subsequently, the most important insights gained from the experimental evaluation are discussed. Section 3.2 focuses on results concerning cost functions, Section 3.3 concerning gradient-based methods and Section 3.4 covers results regarding least-squares methods.

3.1 Benchmarks

To evaluate the properties of policy evaluation methods under various conditions, we selected a number of representative benchmark tasks with different specifications. We computed the algorithms' predictions with an increasing number of training data points, and compared their quality with respect to the MSE, MSBE and MSPBE. These experiments are performed on six different Markov decision processes, three with discrete and three with continuous state space. Most experiments were performed both with on-policy and off-policy samples. We also evaluated different feature representations which altogether resulted in the following 12 settings.

1. 14-State Boyan Chain
2. Baird Star Example
3. 400-State Random MDP On-policy
4. 400-State Random MDP Off-policy
5. Linearized Cart-Pole Balancing On-policy Imperfect Features
6. Linearized Cart-Pole Balancing Off-policy Imperfect Features
7. Linearized Cart-Pole Balancing On-policy Perfect Features
8. Linearized Cart-Pole Balancing Off-policy Perfect Features
9. Cart-Pole Swingup On-policy
10. Cart-Pole Swingup Off-policy

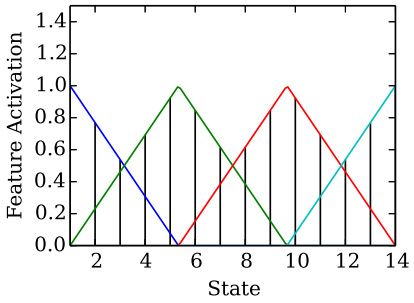


Figure 13: Feature Activation for the Boyan chain benchmark. The state space is densely covered with triangle-shaped basis functions.

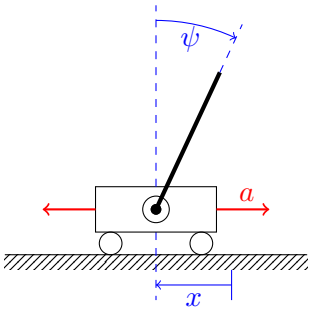


Figure 14: The Cart-Pole System. The pendulum has to be balanced around the peak by moving the cart.

11. 20-link Linearized Pole Balancing On-policy

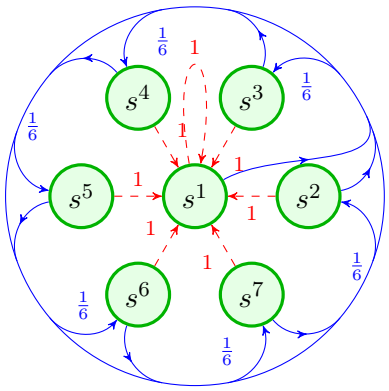
12. 20-link Linearized Pole Balancing Off-policy

3.1.1 Boyan's Chain (Benchmark 1)

The first benchmark MDP is the classic chain example from Boyan (2002). We considered a chain of 14 states $\mathcal{S} = \{s^1, \ldots, s^{14}\}$ and one action. Each transition from state s^i results in state s^{i+1} or s^{i+2} with equal probability and a reward of -3. If the agent is in the second last state s^{13}, it always proceeds to the last state with reward -2 and subsequently stays in this state forever with zero reward. A visualization of a 7-state version of the Boyan chain is given in Figure 6. We chose a discount factor of $\gamma = 0.95$ and four-dimensional feature description with triangular-shaped basis functions covering the state space (Figure 13). The true value function, which is linearly decreasing from s^1 to s^{14}, can be represented perfectly.

3.1.2 Baird's Star Example (Benchmark 2)

Baird's star (Baird, 1995) is a well known example for divergence of TD learning in off-policy scenarios. It is often referred to as "star" MDP as its states can be ordered as a star, one central state and six states at the edges, as shown in Figure 15. There are two



Features:

$$\phi(s^i) = 2e_i + [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^T \quad \text{for } i = 2 \dots 7$$

$$\phi(s^1) = e_1 + 2e_7 = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 2]^T$$

Policies:

$$\pi_B(\cdot\,|s^i) = \begin{cases} \frac{1}{7} & \text{for - -} \\ \frac{6}{7} & \text{for ——} \end{cases}, \quad \text{for } i = 1 \dots 7$$

$$\pi_G(\cdot\,|s^i) = \begin{cases} 1 & \text{for - -} \\ 0 & \text{for ——} \end{cases}, \quad \text{for } i = 1 \dots 7$$

Figure 15: Baird's Star: 7-State Star MDP, a classic off-policy example problem from Baird (1995) in which TD learning diverges for all step-sizes. While the label of a transition denotes its probability, the reward is always zero. The vector e_i denotes the i-th unit vector.

actions. The solid action chooses one of the solid edges with equal probability and the dashed action always chooses the edge to the central state. We set the discount factor $\gamma = 0.99$ and assume zero reward for each transition. Hence, the true value function is zero in every state for all policies. The evaluation policy always takes the dashed action, and hence, goes to the central state. However, the behavior policy, that is, the policy used to generate the samples, chooses the solid action with probability 6/7. The feature vector has eight components. For the outside-states s^i, $i = 2, \dots, 7$, the i-th entry has value 2 and the last entry is 1 (cf. Figure 15). All other entries are zero. The central state sets the first component of $\boldsymbol{\phi}$ to 1 and the last component to 2. All other entries are again zero. We used $\boldsymbol{\theta}_0 = [1\ 1\ 1\ 1\ 1\ 1\ 10\ 1]^T$ as initial parameter vector for the methods that allow specifying a start estimate. Although the true value function is zero everywhere, TD-learning is known to diverge for this initialization of the parameter-vector.

3.1.3 Randomly Sampled MDP (Benchmarks 3 and 4)

To evaluate the prediction in MDPs with more states and of a less constructed nature, we used a randomly generated discrete MDP with 400 states and 10 actions. The transition probabilities were distributed uniformly with a small additive constant to ensure ergodicity of the MDP, that is,

$$\mathcal{P}(s'|a, s) \propto p^a_{ss'} + 10^{-5}, \quad p^a_{ss'} \sim U[0, 1].$$

The data-generating policy, the target policy as well as the start distribution are sampled in a similar manner. The rewards are uniformly distributed, that is, $r(s^i, a^j) \sim U[0, 1]$. Each state is represented by a 201-dimensional feature vector, 200 dimensions which have been generated by sampling from a uniform distribution and one additional constant feature. The MDP, the policies and the features are sampled once and then kept fix throughout all experiments (all independent trials were executed in the same setting). As behavior- and target-policy were generated independently and differ substantially for Benchmark 4,

the algorithms were tested in a difficult off-policy setting. The discount factor is set to $\gamma = 0.95$.

3.1.4 Linearized Cart-Pole-Balancing (Benchmarks 5- 8)

The Cart-Pole Balancing problem is a well known benchmark task which has been used for various reinforcement learning algorithms. As we want to know the perfect feature representation also for a continuous system, we linearized cart-pole dynamics and formulated the task as a linear system with a quadratic reward function and Gaussian noise. Linear-Quadratic-Gaussian (LQG) systems are one of the few continuous settings for which we can compute the true value function exactly. The perfect features for the value function of a LQG system are all first and second order terms of the state vector $\boldsymbol{s}$.

Figure 14 visualizes the physical setting of the benchmark. A pole with mass m and length l is connected to a cart of mass M. It can rotate 360° and the cart can move right and left. The task is to balance the pole upright. The state $\boldsymbol{s} = [\psi, \dot{\psi}, x, \dot{x}]^T$ consists of the angle of the pendulum ψ, its angular velocity $\dot{\psi}$, the cart position x and its velocity $\dot{x}$. The action a acts as a horizontal force on the cart. The system dynamics are given by (cf. Deisenroth, 2010, Appendix C.2 with $\psi = \theta + \pi$)

$$\ddot{\psi} = \frac{-3ml\dot{\psi}^2 \sin(\psi)\cos(\psi) + 6(M+m)g\sin(\psi) - 6(a - b\dot{\psi})\cos(\psi)}{4l(M+m) - 3ml\cos(\psi)} \quad \text{and} \tag{39}$$

$$\ddot{x} = \frac{-2ml\dot{\psi}^2 \sin(\psi) + 3mg\sin(\psi)\cos(\psi) + 4a - 4b\dot{\psi}}{4(M+m) - 3m\cos(\psi)}, \tag{40}$$

where $g = 9.81\frac{m}{s^2}$ and b is a friction coefficient of the cart on the ground (no friction is assumed between pole and cart). If the pole is initialized at the upright position and the policy is keeping the pole around this upright position, the system dynamics can be approximated accurately by linearizing the system at $\psi = 0$. In this case, the linearization yields $\sin\psi \approx \psi$, $\dot{\psi}^2 \approx 0$ and $\cos\psi \approx 1$ and we obtain the linear system

$$\boldsymbol{s}_{t+1} = \begin{bmatrix} \psi_{t+1} \\ \dot{\psi}_{t+1} \\ x_{t+1} \\ \dot{x}_{t+1} \end{bmatrix} = \begin{bmatrix} \psi_t \\ \dot{\psi}_t \\ x_t \\ \dot{x}_t \end{bmatrix} + \Delta t \begin{bmatrix} \dot{\psi}_t \\ \frac{3(M+m)\psi - 3a + 3b\dot{\psi}}{4Ml - ml} \\ \dot{x}_t \\ \frac{3mg\psi + 4a - 4b\dot{\psi}}{4M - m} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ z \end{bmatrix},$$

where the time difference between two transitions is denoted by $\Delta t = 0.1$s and z is Gaussian noise on the velocity of the cart with standard deviation 0.01. We set the length of the pole l to 0.6m, the mass of the cart M to 0.5kg, the mass of the pole m to 0.5kg and the friction coefficient of b to $0.1\,\mathrm{N(ms)}^{-1}$. The reward function is given by

$$R(\boldsymbol{s}, a) = R(\psi, \dot{\psi}, x, \dot{x}, a) = -100\psi^2 - x^2 - \frac{1}{10}a^2,$$

that is, deviations from the desired pole position are strongly penalized, while large offsets of the cart and the magnitude of the current action cause only minor costs. As the transition model is a Gaussian with a linear mean-function and the reward function is quadratic, the exact value function and optimal policy can be computed by dynamic programming (Bertsekas

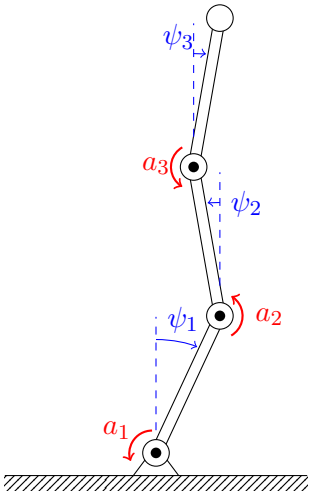


Figure 16: Balancing setup of a 3-link actuated pendulum. Each joint i is actuated by the signal a_i. The state of each joint is denoted by its angle ψ_i against the vertical direction. The pole is supposed to be balanced upright, that is, all ψ_i should be as close to 0 as possible.

and Tsitsiklis, 1996). It is well known that the features of the true value function are given by a constant plus all squared terms of the state[7] $\phi_p(\boldsymbol{s}) = [1, s_1^2, s_1 s_2, s_1, s_3, s_1 s_4, s_2^2, \ldots, s_4^2]^T \in \mathbb{R}^{11}$. The optimal policy $\pi(a|\boldsymbol{s}) = \mathcal{N}(a|\boldsymbol{\beta}^T \boldsymbol{s}, \sigma^2)$ is linear. The target policy π_G is set to the optimal policy, that is, the gains $\boldsymbol{\beta}$ are obtained by dynamic programming and the exploration rate σ^2 is set to a low noise level. The data-generating policy π_B uses the same $\boldsymbol{\beta}$ but a higher noise level in the off-policy case.

To additionally compare the algorithms on a approximate feature representation, we used $\phi_a(\boldsymbol{s}) = [1, s_1^2, s_2^2, s_3^2, s_4^2]^T \in \mathbb{R}^5$ as feature vector in Benchmarks 5 and 6. All evaluations were generated with a discount factor of $\gamma = 0.95$.

3.1.5 Linearized 20-link Balancing (Benchmark 11 and 12)

To evaluate the algorithms on systems with higher-dimensional state- and action-spaces, we considered a 20-link actuated inverted pendulum. Each of the 20 rotational joints are controlled by motor torques $\boldsymbol{a} = [a_1, \ldots a_{20}]^T$ to keep the pendulum balanced upright. See Figure 16 for a visualization of a pendulum with 3 links. The difference in the angle to the upright position of link i is denoted by ψ_i. The state space is 40 dimensional and consists of the angles of each joint $\boldsymbol{q} = [\psi_1, \ldots, \psi_{20}]^T$ and the angular velocities $\dot{\boldsymbol{q}} = [\dot{\psi}_1, \ldots, \dot{\psi}_{20}]^T$.

The derivation of the linearized system dynamics can be found in the supplementary material and yields

$$\begin{bmatrix} \mathbf{q}_{t+1} \\ \dot{\mathbf{q}}_{t+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \Delta t\, \boldsymbol{I} \\ -\Delta t\, \mathbf{M}^{-1}\boldsymbol{U} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \mathbf{q}_t \\ \dot{\mathbf{q}}_t \end{bmatrix} + \Delta t \begin{bmatrix} \mathbf{0} \\ \mathbf{M}^{-1} \end{bmatrix} \mathbf{a} + \boldsymbol{z},$$

7. The linear terms disappear as we have linearized at $\boldsymbol{s} = \mathbf{0}$.

where $\Delta t = 0.1$s is the time difference of two time steps and $\boldsymbol{M}$ is the mass matrix in the upright position. Its entries are computed by $\boldsymbol{M}_{ih} = l^2(21 - \max(i,h))m$ with length $l = 5$m and mass $m = 1$kg of each link. The matrix $\boldsymbol{U}$ is a diagonal matrix with entries $\boldsymbol{U}_{ii} = -gl(21 - i)m$. Each component of $\boldsymbol{z}$ contains Gaussian noise. Again, we used a quadratic reward function

$$R(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{a}) = -\boldsymbol{q}^T \boldsymbol{q},$$

which penalizes deviations from the upright position. The target policy is given by the optimal policy (obtained by dynamic programming) with Gaussian noise and analogously the behavior policy but with increased noise level for the off-policy estimation case. A discount factor of $\gamma = 0.95$ and a 41-dimensional approximate feature representation $\phi(\boldsymbol{q}, \dot{\boldsymbol{q}}) = [\psi_1^2, \psi_2^2, \ldots, \psi_{20}^2, \dot{\psi}_1^2, \dot{\psi}_2^2, \ldots, \dot{\psi}_{20}^2, 1]^T$ were used in the experiments.

3.1.6 Cart-Pole Swing-up (Benchmarks 9 and 10)

Besides discrete and linear systems, we also include a non-linear problem, the cart-pole swing-up task. The non-linear system dynamics from Equations (39) and (40) were used and the constants were set to the same values as in the linearized task. The reward function directly rewards the current height of the pole and mildly penalizes offsets of the cart

$$R(\boldsymbol{s}, a) = R(\psi, \dot{\psi}, x, \dot{x}, a) = \cos(\psi) - 10^{-5}|x|.$$

We used an approximately optimal policy learned with the PILCO-Framework (Deisenroth and Rasmussen, 2011) and added Gaussian noise to each action a. The resulting policy manages to swing-up and balance the pendulum in about 70% of the trials, depending on the initial pole position which is sampled uniformly. Each episode consists of 200 timesteps of 0.15s duration. A normalized radial basis function network and an additional constant feature has been chosen as feature representation. To obtain a compact representation, we first covered the four-dimensional state space with a grid of basis functions and then removed all features for which the summed activations were below a certain threshold. Thus, we omitted unused basis functions which are located in areas of the state space, which are not visited. The resulting feature vector had 295 dimensions.

3.1.7 Hyper-Parameter Optimization

The behavior of policy evaluation methods can be influenced by adjusting their hyper-parameters. We set those parameters by performing an exhaustive grid-search in the hyper-parameter space minimizing the MSBE (for the residual-gradient algorithm and BRM) or MSPBE. Optimizing for MSBE or MSPBE introduces a slight bias in the choice of the optimal parameters. For example, smaller value of λ for the eligibility traces are preferred as small values of λ as the objective bias is not taken into account. However, as opposed to the MSE, these objectives can be computed without knowledge of the true values, and, hence, can be evaluated also in practice on a small set of samples. We evaluated the algorithms for an increasing number of observed time steps and computed a weighted average over the errors of all considered time steps to obtain a single score per trial. We increased the weights from 1 for the first to 2 for the last estimate and therefore put emphasis on a good final value of the estimates but also promoted fast convergence. The scores of three independent trials

Parameter	Evaluated Values
α	$2 \cdot 10^{-4}$, $5 \cdot 10^{-4}, 10^{-3}$, $0.002, \ldots, 0.009, 0.01,$ $0.02, \ldots, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5$
α_{LSPE}	$0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 0.8, 0.9, 1$
α_{FPKF}	$0.01, 0.1, 0.3, 0.5, 0.8, 1$
β_{FPKF}	$1, 10, 100, 1000,$
τ_{FPKF}	$0, 500, 1000$
μ	$10^{-4}, 10^{-3}, 10^{-2}, 0.05, 0.1, 0.5, 1, 2, 4, 8, 16$
λ	$0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1$
ϵ	$10^5, 10^3, 10^2, 10, 1, 0.1, 0.01$
ζ	$0.01, 0.02, \ldots 0.09, 0.1, 0.2, \ldots, 0.9, 1, 5, 10, 30$
η	$0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.5$

Table 3: Considered values in the grid-search parameter optimization for the algorithms listed in Table 4.

were averaged to obtain a more stable cost function during hyper-parameter grid-search. Table 4 provides a listing of all considered algorithms with their hyper-parameters. Each parameter in Table 4 is evaluated in the grid-search at the values listed in Table 3.

All shown results are averages over 50 independent trials for continuous MDPs or 200 trials for discrete MDPs, if not stated otherwise. For discrete and linear continuous systems, the MSBE / MSPBE / MSE values were calculated exactly, whereas the stationary state distribution d^π is approximated by samples. For the non-linear continuous system, also the expectations inside the MSBE and MSPBE were approximated by samples while the true value function was estimated by exhaustive Monte-Carlo roll-outs.[8] We often used the square-root of costs to present results in the following, which are denoted by RMSE, RMSBE or RMSPBE.

3.1.8 Normalization of Features

Two types of features were used in the continuous environments, the squared terms of the state $\boldsymbol{s}$ in the linear case, and a radial basis function network in the non-linear case. Both representations were normalized. For the squared terms, we subtracted the mean feature vector and divided by the standard deviation of the individual features. Hence, each feature was normalized to have zero-mean and unit variance. For the radial basis function representation, we always divided the activations of the radial basis functions by their sum,

8. Ten roll-outs were used for each sample of the stationary distribution. We compared the Monte-Carlo estimates from ten roll-outs against estimates from 20 roll-outs on a small subset of samples but did not observe significant differences. Due to the high computational effort, we therefore settled for ten roll-outs per state.

(A) residual-gradient (RG) algorithm
- constant step-sizes $\alpha_t = \alpha$

(B) RG algorithm with double samples (RG DS)
- constant step-sizes $\alpha_t = \alpha$

(C) TD(λ) learning
- constant step-sizes $\alpha_t = \alpha$
- bootstrapping trade-off λ

(D) TD learning decreasing steps
- diminishing step-sizes $\alpha_t = \zeta t^{-\eta}$

(E) GTD
- constant step-sizes $\alpha_t = \alpha$
- second step-size $\beta_t = \alpha\mu$

(F) GTD2
- constant step-sizes $\alpha_t = \alpha$
- second step-size $\beta_t = \alpha\mu$

(G) TDC(λ)
- constant step-sizes $\alpha_t = \alpha$
- second estimate step-size $\beta_t = \alpha\mu$
- bootstrapping trade-off λ

(H) LSTD(λ)
- ℓ_2 regularization ϵ
- bootstrapping trade-off λ

(I) LSTD(λ)-TO
- ℓ_2 regularization ϵ
- bootstrapping trade-off λ

(J) LSPE(λ)
- constant step-sizes $\alpha_t = \alpha_{\text{LSPE}}$
- bootstrapping trade-off λ

(K) LSPE(λ)-TO
- constant step-sizes $\alpha_t = \alpha_{\text{LSPE}}$
- bootstrapping trade-off λ

(L) FPKF(λ)
- constant step-sizes $\alpha_t = \begin{cases} \alpha_{\text{FPKF}} \frac{\beta_{\text{FPKF}}}{\beta_{\text{FPKF}}+t} & \text{for } \tau_{\text{FPKF}} \leq t \\ 0 \quad \text{otherwise} \end{cases}$
- bootstrapping trade-off λ

(M) BRM
- ℓ_2 regularization ϵ
- bootstrapping trade-off λ

(N) BRM with double samples (BRM DS)
- ℓ_2 regularization ϵ

Table 4: Overview of all considered algorithms with their hyper-parameters. As GPTD(λ) is equivalent to LSTD(λ) with ℓ_2 regularization, it is not included explicitly.

that is, the sum of their activations is always 1. Since the feature function includes an additional constant feature 1, the total activation for each state $\boldsymbol{s}$ is $\|\phi(\boldsymbol{s})\|_1 = 2$. Features in discrete settings were not normalized.

3.2 Insights on the Algorithms-Defining Cost Functions

The objective function of the policy evaluation algorithm determines its fixpoint, and, hence, largely influences the final quality of the predictions. As discussed in Section 2.1, only few theoretical results such as loose bounds could be derived. Additionally, constructed examples show that the quality of fixpoints with respect to the mean-squared error highly depends on the problem setting. Therefore, empirical comparisons of the MSTDE, MSBE and MSPBE fixpoints for common problems with different challenges are of particular interest.

Message 1 *Empirically, the magnitudes of the biases of different objective functions with respect to the MSE fixpoint are: bias(MSTDE) $\geq$ bias(MSBE) $\geq$ bias(MSPBE).*

	bias of		
	MSTDE	MSBE	MSPBE
1. 14-State Boyan Chain	1.93	**0.06**	0.10
2. Baird Star Example	**0.00**	0.00	0.03
3. 400-State Random MDP On-policy	0.06	**0.04**	**0.04**
4. 400-State Random MDP Off-policy	0.06	0.08	**0.05**
5. Lin. Cart-Pole Balancing On-pol. Imp. Feat.	4.52	3.80	**2.60**
6. Lin. Cart-Pole Balancing Off-pol. Imp. Feat.	4.37	3.82	**2.47**
7. Lin. Cart-Pole Balancing On-pol. Perf. Feat.	1.92	0.05	**0.03**
8. Lin. Cart-Pole Balancing Off-pol. Perf. Feat.	1.94	0.13	**0.04**
9. Cart-Pole Swingup On-policy	3.83	3.82	**1.99**
10. Cart-Pole Swingup Off-policy	4.28	4.30	**2.17**
11. 20-link Lin. Pole Balancing On-pol.	7.71	7.45	**4.27**
12. 20-link Lin. Pole Balancing Off-pol.	0.08	0.08	**0.04**

Table 5: Mean squared error values of fixpoints of other cost-functions: The fixpoints are estimated by the prediction of LSTD, BRM or BRM with double samples after convergence. The MSPBE has the lowest bias in almost all experiments.

Table 5 shows the observed MSE value of each fixpoint for every benchmark problem. We estimated the MSBE, MSTDE and MSPBE fixpoints by running either the Bellman residual minimization algorithm with or without double sampling or LSTD until convergence (up to a certain accuracy, without eligibility traces). While this procedure introduces approximation errors, which are not entirely neglectable, it still allows us to compare the fixpoints of the cost functions. We ensured that regularization did not impair with the results by comparing the fixpoint estimations for different regularization parameters.

The results confirm the findings of Scherrer (2010) on discrete MDPs. The MSPBE fixpoint yields a lower MSE than the MSBE in all continuous experiments. MSTDE and MSBE are observed to generate substantially inferior predictions, often with errors almost twice as big. While the MSTDE usually yields the worst predictions, the difference to the MSBE depends on the amount of stochasticity in each transition. For example, both are almost identical on the swing-up task due to low noise in the policy and the MDP. While the MSPBE and MSBE fixpoints are identical to the MSE fixpoint for experiments with perfect feature representations (up to numerical issues, Benchmarks 1,2,7,8), the MSTDE fixpoint is often substantially different (Benchmarks 1,7,8). The problem of a potential dramatic failure of the MSPBE solution, as sketched by Scherrer (2010), was not encountered throughout all evaluations.

Message 2 *Optimizing for the MSBE instead of MSTDE by using double samples introduces high variance in the estimate. Particularly, Bellman residual minimization requires stronger regularization which results in slower convergence than relying on one sample per transition.*

The objective function does not only determine the objective bias but also affects the sampling error (see Figure 10). Using double samples, that is, optimizing for MSBE instead of MSTDE, decreases the objective bias, however, our experiments show that the second

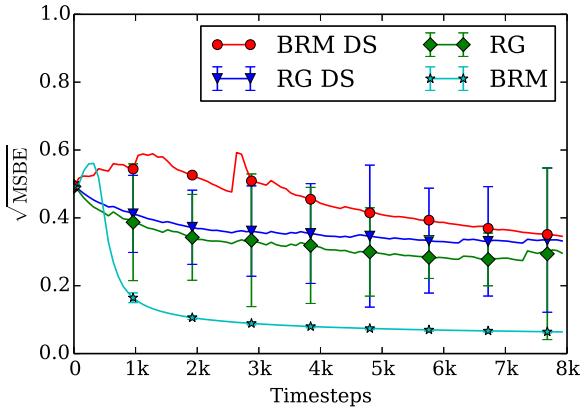


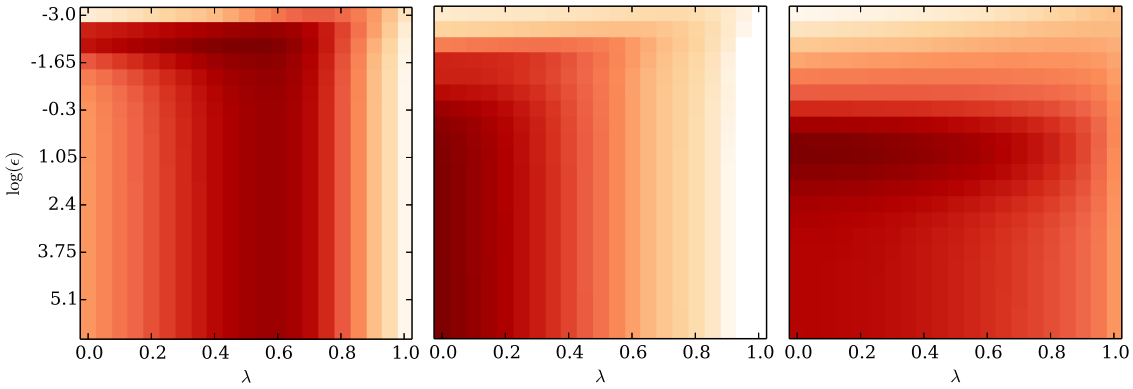
Figure 17: Comparison of double-sampling for BRM and the residual-gradient algorithm in systems with high variance (4. 400-State Random MDP Off-policy). The error bars indicating standard deviation of BRM with double-sampling (BRM DS) are omitted for clarity.

sample per transition is not the only price to pay. To determine whether the sampling error for the two objectives is different, we compared the online performance of residual-gradient algorithm (RG) and Bellman residual minimization (BRM) with or without double sampling (DS). We have observed that double-sampling variants converge significantly slower, that is, their predictions have higher variance. The effect is particularly present in MDPs with high variance such as the random discrete MDPs, see Figure 17. Double-sampling algorithms require stronger regularization and therefore converge slower. Bellman residual minimization suffers more from this effect than the residual-gradient algorithm.

Message 3 *Interpolating between the MSPBE/MSTDE and the MSE with eligibility traces can improve the performance of policy evaluation.*

In Example 2 in Section 2.4.1, we illustrated the benefits of eligibility traces with a specially tailored MDP for which we could control its stochasticity easily. The question remains whether the interpolation between the MSE and MSPBE is also useful for noise levels in MDPs encountered in practice. Is the noise so large that the variance is always the dominant source of error or does reducing the bias with eligibility traces pays off? We therefore compared the MSE of LSTD(λ) and TD(λ) predictions for different λ values on several benchmark tasks. Representative results of LSTD, shown in Figure 18b, confirm that eligibility traces are of no use if no bias is present in the MSPBE due to perfect features. The same holds for systems with large stochasticity such as in the randomly sampled discrete MDP shown in Figure 18c. Yet, interpolating between the MSPBE and MSE boosts the performance significantly if the MSPBE introduces a bias due to an imperfect feature representation and the variance of the MDP is not too high. Such behavior is shown in Figure 18a, where we used the approximate features instead of the perfect feature representation.

Similar to LSTD, TD learning can be improved by eligibility traces as shown for the linearized cart pole balancing benchmark with imperfect features in Figure 19a. Best pre-



(a) Cart-Pole Balancing Imp. Feat. Off-Policy (6.)

(b) Pole Balancing Perf. Feat. Off-Policy (8.)

(c) Discrete Random MDP On-Policy (3.)

Figure 18: Hyper-parameter space of LSTD(λ). Each point is the logarithm of the averaged MSE. Darker colors denote low errors, while white indicates divergence. The colormaps are log log-normalized, that is, the absolute difference in the dark regions are smaller than those in the bright areas. The regularization parameter ϵ is plotted logarithmically on the vertical axis and the eligibility traces parameter λ on the horizontal one.

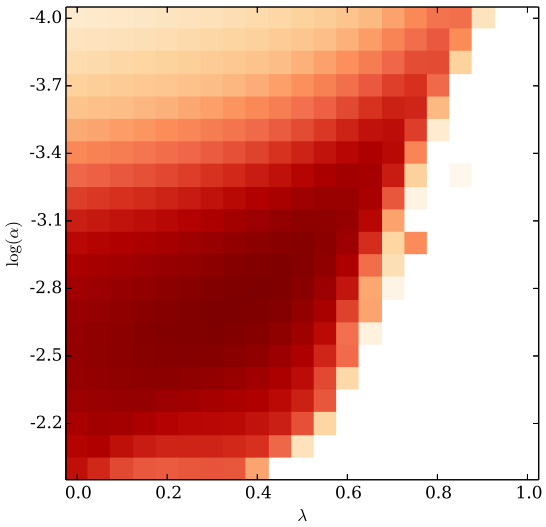
dictions are obtained with $0.1 < \lambda < 0.5$ depending on the step-size α. Yet, different to LSTD, TD learning also benefits from eligibility traces for perfect features (see Figure 19b). They speed up learning by reducing the optimization error of gradient-based approaches (cf. Figure 10) and make the algorithms more robust to the choice of the step-size. These benefits are also present in systems with high stochasticity as Figure 19c indicates. While eligibility traces diminishes the prediction quality of LSTD in such highly stochastic systems, TD learning works best for all λ settings as long as the step-size is set appropriately.

3.2.1 Double Sampling vs. Eligibility Traces

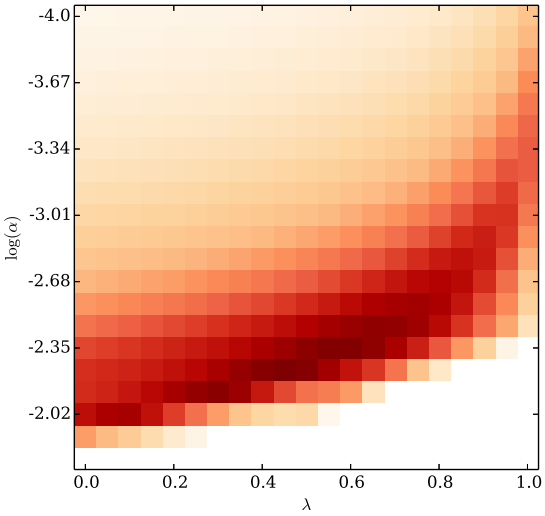
Both, double sampling and eligibility-traces, can be used to reduce the bias of the MSTDE objective at the price of higher variance. However, which approach works better in practice? To shed some light on this matter, we compared BRM with and without double sampling and BRM with eligibility traces. The results for the cart-pole balancing task with imperfect features (Benchmark 5) are shown in Figure 20. We chose the λ-parameter such that both approaches have the same convergence speed, that is, comparable variance. The plot shows that eligibility traces can reduce the bias of the MSTDE more than double sampling at the same increase of variance.

3.2.2 MSPBE vs. MSE Performance

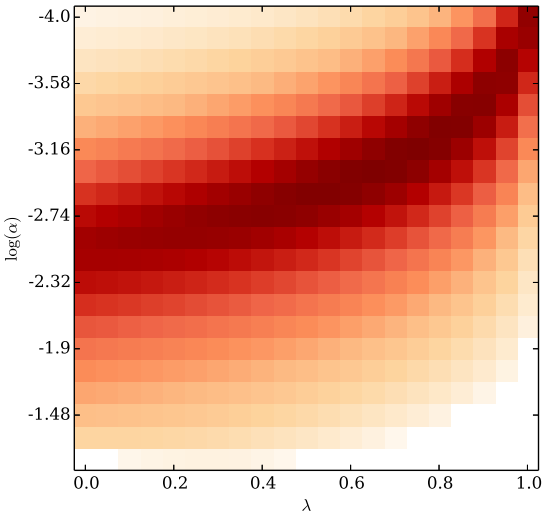
During our evaluation, we often made the interesting observation that a prediction with significantly lower MSPBE than another prediction does not necessarily have a significantly lower MSE. See Figure 21 for such an example, which shows the MSE and MSPBE of predictions for the randomly sampled discrete MDP (Benchmark 3). The performance of



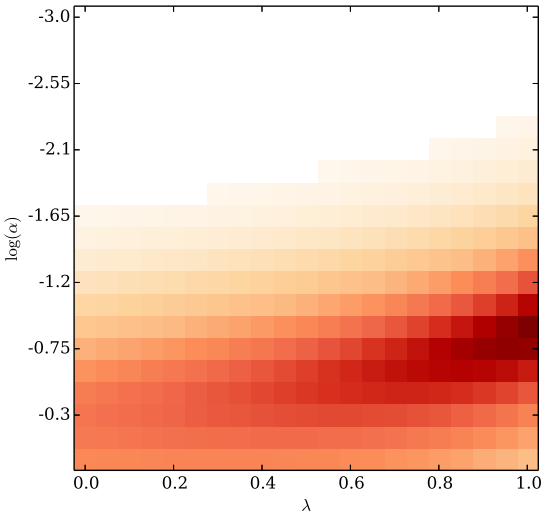
(a) Cart-Pole Balancing Imperf. Feat. Off-Policy (6)



(b) Cart-Pole Balancing Perf. Feat. On-Policy (7)



(c) Discrete Random MDP On-Policy (3)



(d) Boyan Chain (1)

Figure 19: Hyper-parameter space of TD(λ). The color of each point represents the averaged MSE. Darker colors are denoted to low errors, while white indicates divergence. The colormaps are log log-normalized, that is, the absolute difference in the dark regions are smaller than those in the bright areas.

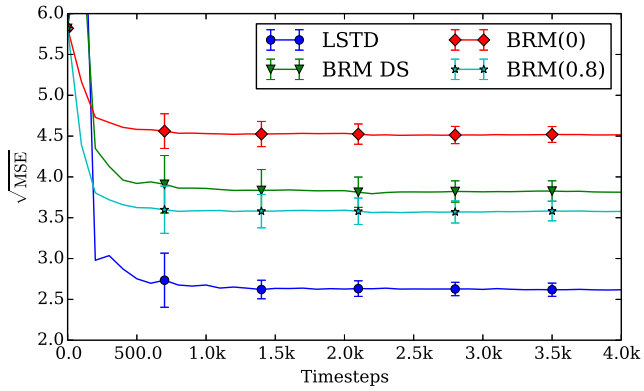


Figure 20: Comparison of bias reduction with double-sampling or eligibility traces for BRM. Eligibility traces introduce less or equal variance than double-sampling and decrease the bias more than double-sampling for linearized cart-pole balancing with imperfect features. Still, LSTD produces less biased predictions at the same level of variance.

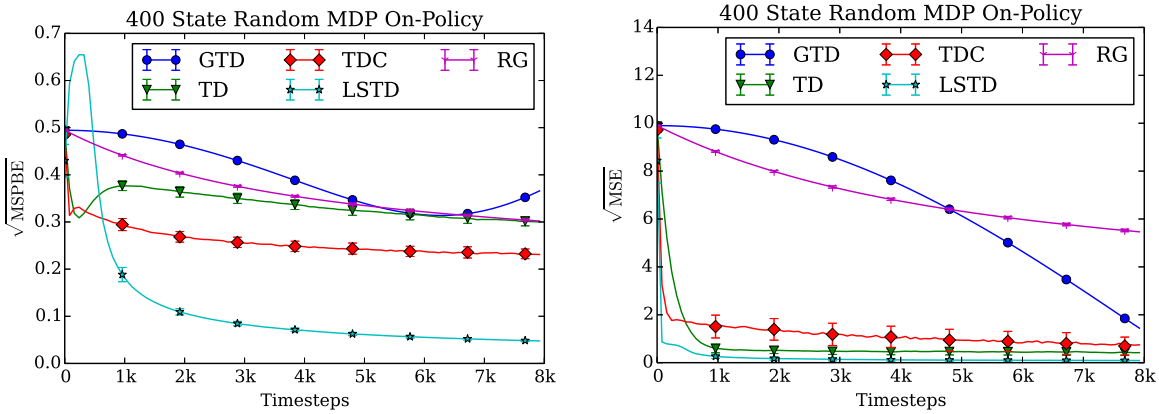


Figure 21: Difference between MSE- and MSPBE-values of the same predictions on the random discrete MDP. Differences w.r.t. MSPBE are not always present in the MSE (see the RG performance).

LSTD and TDC or TD is very different with respect to the MSPBE but they perform almost identically w.r.t. the MSE. While this observation does not always hold (compare for example RG and LSTD), we experienced similar effects in many experiments including continuous MDPs.

3.3 Results on Gradient-based Methods

In this section, we present the most important observations for gradient-based methods.

Message 4 *Normalization of the features is crucial for the prediction quality of gradient-based temporal-difference methods.*

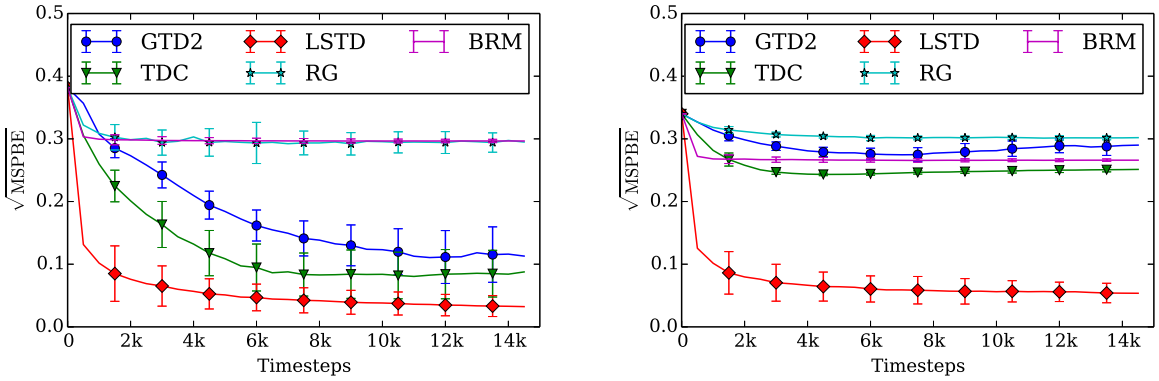
Throughout all experiments, we observed that normalizing the feature representation improves, or least does not harm, the performance of all temporal-difference methods. However, for gradient-based approaches, feature normalization is crucial for a good performance. Features can be normalized *per time step*, for example, all components of the feature vector $\boldsymbol{\phi}_t$ sum up to one, or *per dimension*, for example, each feature is shifted and scaled such that it has mean zero and variance one. Per-time-step normalization is, for example, typically used in radial basis function networks (see Benchmark 11 and 12) to ensure that each time step has the same magnitude of activation and consequently all transition samples have the same weight. As such, its effect resembles that of using natural gradients (Amari, 1998; a discussion of the relation between using the Hessian and natural gradients is provided by Roux and Fitzgibbon, 2010). Since the gradient varies less for different states with per-time-step normalization, the actual distribution of states is less important and the estimate becomes more robust for finitely many samples.

Per-dimension normalization gives each feature comparable importance. Under the assumption that the value function changes similarly fast in each feature dimension, per-dimension normalization causes the Hessian matrix to become more isotropic. It has therefore an effect similar to using the inverse of the Hessian to adjust the gradient as least-squares methods do. However, least-squares methods still can benefit from such a normalization since their regularization has more equate effect on all dimensions.

We compared per-dimension normalized (Figure 22a) and unnormalized features (Figure 22b) for the cart-pole balancing task. The results show that the performance of gradient-based approaches degrade drastically without normalization. As this benchmark requires only little regularization, the performance of least-squared methods is not significantly affected by feature normalization. To understand why normalization plays such an important role for gradient-based methods, consider the MSPBE function in an unnormalized feature space. It may correspond to a quadratic loss function which is flat along some dimension and steep in others, that is, its Hessian contains large and small eigenvalues. Hence, the optimal step-size for gradient descent algorithms can vary significantly per dimension, resulting in either slow convergence of gradient-based algorithms with small step-sizes or a bias if larger step-sizes are used, see, for example, TDC in Figure 22b.

Message 5 *GTD performs worse than its successors GTD2 and TDC. TDC minimizes the MSPBE faster than the other gradient-based algorithms GTD, GTD2 and TD learning.*

We assessed the ability of the gradient based methods TD learning, GTD, GTD2 and TDC to minimize the MSPBE. The results are given in Table 6. Each entry corresponds to the accumulated $\sqrt{\text{MSPBE}}$-values of the predictions for all time steps. Low numbers indicate accurate estimates with small number of samples. We observe that the performance of GTD is always worse than the performance of the other methods except for the Boyan chain (Benchmark 1) and the 20-link Pole Balancing Off-policy task (Benchmark 12). GTD2 converged very slowly in these two experiments. Throughout all tasks, GTD performs significantly worse than other approaches and yields unreliable results in general, that is, sometimes the



(a) Normalized Features.

(b) Unnormalized Features.

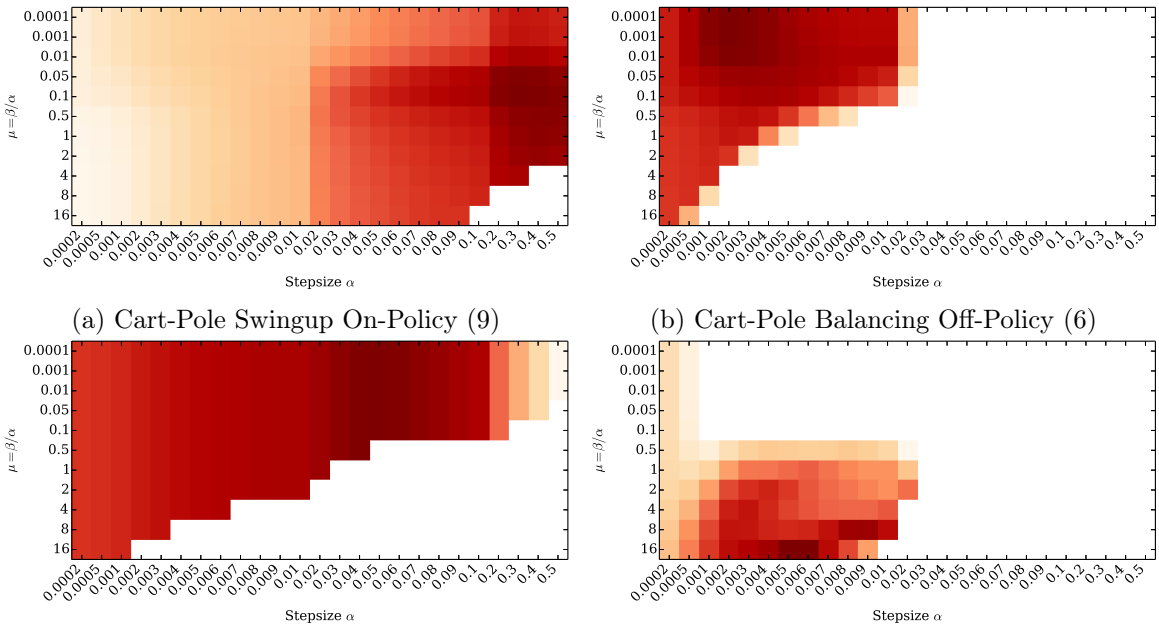
Figure 22: Comparison for the cart pole balancing task (Benchmark 5) with normalized and unnormalized features. Differences in the magnitude of features are particularly harmful for gradient-based approaches.

	GTD	GTD2	TDC	TD
1. 14-State Boyan Chain	58.11	48.65	**16.51**	16.51
2. Baird Star Example	1504.38	1520.09	**1237.70**	$> 10^{10}$
3. 400-State Random MDP On-policy	39.58	31.06	**25.90**	33.62
4. 400-State Random MDP Off-policy	40.90	38.08	**30.50**	37.08
5. Lin. Cart-Pole Balancing On-pol. Imp. Feat.	8.56	5.37	**3.84**	3.84
6. Lin. Cart-Pole Balancing Off-pol. Imp. Feat.	35.86	21.05	**13.31**	13.31
7. Lin. Cart-Pole Balancing On-pol. Perf. Feat.	10.18	8.07	**7.07**	7.98
8. Lin. Cart-Pole Balancing Off-pol. Perf. Feat.	20.65	18.40	**15.47**	19.68
9. Cart-Pole Swingup On-policy	40.21	24.66	**23.08**	25.60
10. Cart-Pole Swingup Off-policy	41.09	30.44	**25.28**	30.14
11. 20-link Lin. Pole Balancing On-pol.	21.97	20.24	**17.22**	18.58
12. 20-link Lin. Pole Balancing Off-pol.	0.39	0.43	**0.26**	0.30

Table 6: Sum of square root MSPBE for all timesteps of GTD, GTD2, TDC and TD learning (TD). GTD is observed to always yield the largest error except for Benchmark 2 and 12. TDC outperformed the other methods in all experiments. The values are obtained after optimizing the hyper-parameters for the individual algorithms.

estimates have a drastically higher error (Benchmark 1, 5, 6). TDC outperformed all other gradient-based methods in minimizing the MSPBE throughout all tasks.

Message 6 *If we optimize the hyper-parameters of TDC, TDC is always at least as good as TD-learning, but comes at the price of optimizing an additional hyper-parameter. Often, hyper-parameter optimization yields very small values for the second learning rate β, in which case TDC reduces to TD-learning.*



(a) Cart-Pole Swingup On-Policy (9)

(b) Cart-Pole Balancing Off-Policy (6)

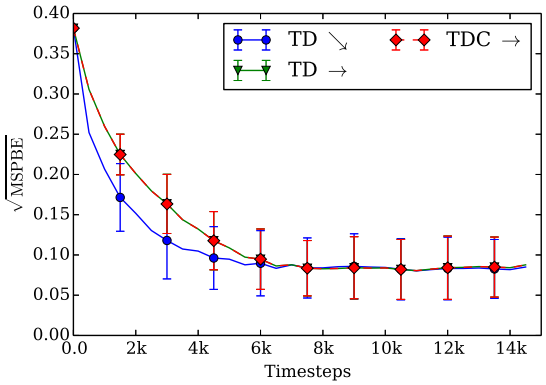
(c) 20-link Pole Balancing Off-Policy (12)

(d) Baird's Star Example (2)

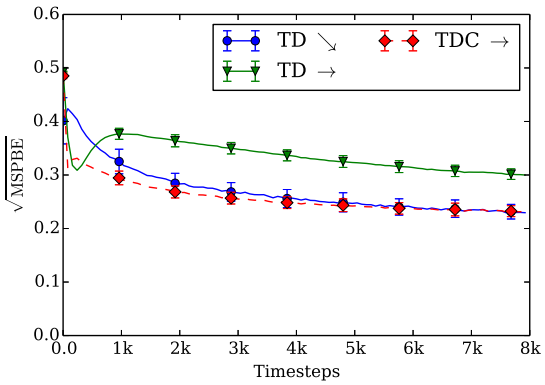
Figure 23: Hyper-parameter space of TDC for $\lambda = 0$. The primary step-size of TDC is denoted by α and $\mu = \beta/\alpha$ is the ratio of secondary to primary step-size. Each point is the logarithm of the averaged MSPBE. Darker colors are denoted to low errors, while white indicates divergence.

As TDC is identical to TD if we set the second learning rate β for the vector $\boldsymbol{w}$ (or the ratio $\mu = \beta/\alpha$) to zero, the performance of TDC is at least as good as that of TD if the hyper-parameters are optimized. As the results in Table 6 show, the difference of TDC and TD is negligible in some tasks (Tasks 1, 5, 6). In these cases, the optimal values for the ratio μ are very small, as the results of the grid-search in Figure 23b indicate, and TDC reduces to TD.

Large μ values were only observed to yield good performance for the Baird's Star task (Benchmark 2) where TD learning always diverged (see Figure 23d). Apart from this example, which is specifically tailored to show divergence of TD learning, TD converged in all off-policy benchmarks. However, even if TD learning converges, the use of the second step-size can be beneficial in some scenarios (e.g., in Benchmark 3, 4, 8, 9 and 10), as the MSPBE of the predictions can be reduced significantly. The grid search results of the Swing-up task shown in Figure 23a as well as the prediction error over time shown in Figure 24b clearly indicate an advantage of TDC. However, TDC comes at the price that we need to optimize the second learning rate as an additional hyper-parameter despite that it may have almost no effect in some problems (see Figure 23c).



(a) Cart-pole Balancing With Imperfect Features, On-policy. (Benchmark 6). The graphs of TDC and TD with constant step-sizes are identical.



(b) Discrete Random MDP, On-policy (Benchmark 3)

Figure 24: Convergence Speed for TD learning with decreasing (TD ↘) and constant step-sizes (TD →) and TDC with constant step-sizes (TDC →).

3.3.1 Constant vs. Decreasing Learning Rates

We also evaluated whether using a decreasing learning rate—an assumption on which the convergence proofs of stochastic gradient-based methods rely—improves the prediction performance for a finite number of time steps. We compared constant learning rates against exponentially decreasing ones (see C and D in Table 4) for TD learning. In most tasks, no significant improvements with decreasing rates could be observed. Only for the cart pole balancing with imperfect features (Benchmark 6) and the discrete random MDP (Benchmark 3), we could speed up the convergence to low-error predictions. Figure 24 illustrates the difference. However, using decreasing learning rates is harder as at least two parameters per learning rate need to be optimized, which we experienced to have high influence on the prediction quality, and, hence, we do not recommend to use decreasing step-sizes for a limited number of observations.

3.3.2 Influence of Hyper-Parameters

We can consider the prediction error of each method as a function of the method's hyper-parameters. As Figure 19 and Figure 23 indicate, these functions are smooth, uni-modal and often even convex for gradient-based algorithms.[9] Only parts of the hyper-parameter spaces are shown, yet, we observed the functions to be well-behaved in general, and, hence, local optimization strategies for selecting hyper-parameters can be employed successfully.

3.4 Results on Least-Squares Methods

In this section, we present the most important insights from the experimental evaluation of least-squares methods.

9. The plots in Figure 19 seem non-convex due to the log-scale of the step-size parameter α.

Task	GTD	GTD2	TD	TDC	RG	RG DS	BRM	BRM DS	LSPE	LSTD	FPKF
1	7.22	6.89	5.56	0.40	5.56	6.83	2.32	0.26	**0.10**	**0.10**	0.79
2	1.74	1.63	0.03	0.03	1.56	2.20	**0.00**	0.00	0.03	0.03	0.22
3	1.44	1.36	1.05	0.75	5.46	2.32	0.12	1.44	**0.09**	**0.09**	4.34
4	1.39	1.97	0.93	1.29	3.10	2.95	**0.10**	3.20	0.13	0.13	9.72
5	2.37	**2.37**	3.58	2.51	4.42	3.75	4.52	3.80	2.60	2.60	2.58
6	2.59	**2.33**	4.37	2.44	4.42	3.88	4.37	3.82	2.47	2.47	2.91
7	5.45	3.12	**0.15**	1.75	3.14	1.19	**0.15**	**0.15**	**0.15**	**0.15**	0.24
8	5.43	4.04	1.95	2.10	3.18	1.53	1.95	1.95	**0.17**	**0.17**	3.82
9	5.13	3.98	3.83	3.86	4.61	4.60	3.83	3.82	**1.97**	1.99	2.88
10	5.45	4.85	4.28	3.91	4.71	4.71	4.28	4.30	4.68	**2.17**	4.28
11	4.29	4.41	7.71	4.75	7.60	7.44	7.71	7.45	**4.26**	4.27	7.30
12	0.058	0.08	0.077	0.052	0.077	0.075	0.077	0.076	0.043	**0.042**	0.081

Table 7: Mean squared errors of final predictions. Task names and descriptions associated with the numbers can be found in Section 3.1. LSPE and LSTD are shown with transition-based off-policy reweighting (LSPE-TO and LSTD-TO).

Message 7 *In general, LSTD and LSPE produce the predictions with lowest errors for sufficiently many observations.*

Table 7 shows the square roots of mean-squared errors ($\sqrt{\text{MSE}}$) of the estimate from each method at the last timestep. The values are the final errors obtained with specific methods for each benchmark. The best prediction is generated either by LSTD or LSPE for almost all tasks. In cases where LSPE has the lowest error, LSTD is only marginally worse and does not depend on a learning rate α to be optimized.

The Star Example of Baird has a true value function of constant $\mathbf{0}$ and is therefore not suited to compare least-squares methods. Each least-squares method can yield perfect results with overly strong regularization. The small difference between the errors of BRM and LSTD in the second row of Table 7 are caused by numerical issues avoidable by stronger regularization.

Interestingly, BRM outperforms all other methods in Benchmark 4, the randomly generated discrete MDP with off-policy samples. The MSTDE has a high bias but at the same time a low variance which seems to be particularly advantageous here as this benchmark is highly stochastic. We experienced unexpected results for the cart-pole balancing tasks with imperfect features (Tasks 5 and 6). Here, the gradient-based approaches perform exceedingly well and GTD even obtains the lowest final MSE value. However, Figure 25 reveals that the reason for this effect is only an artifact of the optimization error introduced by gradient methods. The two sources of error, objective bias and optimization error, counterbalance each other in this example. The MSPBE fixpoint has an error of about 2.5 and LSTD converges to it quickly. The gradient methods, however, converge slower due to their optimization error. Yet, when they approach the MSPBE fixpoint, the estimates pass through regions with lower MSE. As GTD has not converged for the maximum number of evaluated observations, it is still in the region with lower MSE. It therefore yields the best

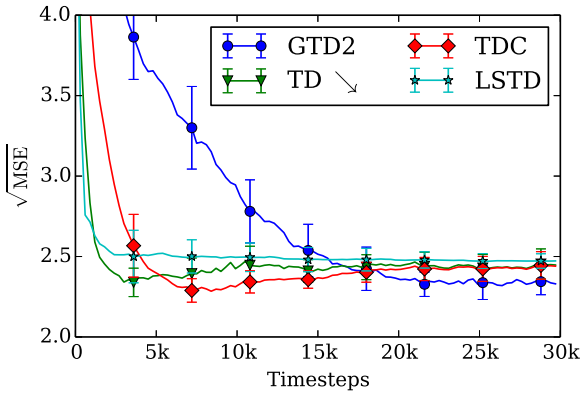


Figure 25: Pole Balancing task with impoverished features. Slightly sub-optimal prediction with respect to the MSPBE yield lowest MSE.

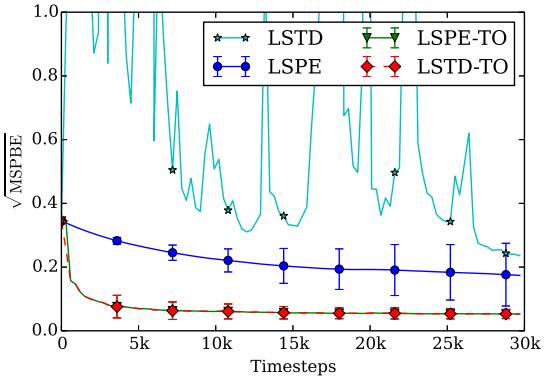
final prediction. However, it would eventually converge to the worse MSPBE fixpoint. Unfortunately, we typically do not have knowledge when such a coincidence happens without actually evaluating the MSE, and, thus, can not exploit this effect. Apart from Tasks 4, 5 and 6, LSTD always yields the lowest or almost the lowest errors, while other methods perform significantly worse on at least some benchmarks (e.g., Task 10 for LSPE). According to the results of our experiments, LSTD is a very accurate and reliable method. It is the method of our choice, although it may behave unstable in some cases, for example, when the number of observations is less than the number of features or some features are redundant, that is, linearly dependent.

Message 8 *In practice, LSTD and LSPE perform well with off-policy samples only if the newly introduced transition reweighting (proposed in Section 2.4.2) is used. The variance of LSTD with standard reweighting makes the algorithm unusable in practice.*

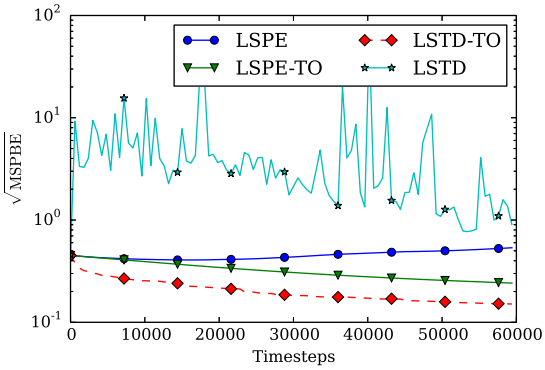
In Section 2.4.2, we proposed an off-policy reweighting based on the entire transition as an alternative to the standard off-policy sample reweighting for LSTD and LSPE. Both reweighting strategies converge to the same solution for infinitely many observations. However, transition reweighting yields much faster convergence as Figure 26 illustrates. Figure 26a compares the reweighting approaches on the linearized cart-pole problem (Benchmark 6). Despite strong ℓ_2-regularization, LSTD yields very noisy estimates with standard reweighting, rendering the algorithm inapplicable. The variance induced by the standard reweighting prevents fast convergence, as the variance of 1.0 between different experiment runs indicates. While the estimates of LSPE with standard reweighting show decreasing error over time (due to a very small step size chosen by the hyper-parameter optimization), the increasing standard deviation indicates that the variance of the estimates is problematic. In contrast, LSPE and LSTD with transition reweighting converge substantially faster and yield good estimates after already 5,000 time steps. The benefit of transition reweighting is even more salient in the results of the off-policy cart-pole swing-up task (Benchmark 10) shown in Figure 26b on a logarithmic scale. The observations are consistent with all off-policy tasks (see Table 8). The price for using off-policy samples instead of on-policy samples, in terms of convergence

	LSPE	LSPE-TO	LSTD	LSTD-TO
4. 400-State Random MDP Off-policy	110.16	21.30	1727.10	**15.50**
6. Lin. Cart-Pole Balancing Off-pol. Imp. Feat.	22.08	6.88	223.64	**6.79**
8. Lin. Cart-Pole Balancing Off-pol. Perf. Feat.	15.52	4.38	49.27	**3.52**
10. Cart-Pole Swingup Off-policy	45.26	32.11	493.18	**20.58**
12. 20-link Lin. Pole Balancing Off-pol.	0.43	**0.24**	0.43	0.32

Table 8: Sum of square-roots of MSPBE for all timesteps of LSPE and LSTD with standard importance reweighting and transition off-policy reweighting (LSPE-TO, LSTD-TO). The Baird-Star Example (Task 2) is omitted as it is not suited well for evaluating least-squares approaches since perfect estimates can be achieved with overly strong regularization.



(a) Cart pole balancing with 5 features, off-policy (Benchmark 6). The error-bars of LSTD with standard reweighting are omitted for visibility.



(b) Cart pole swingup, off-policy (Benchmark 10) on a logarithmic scale.

Figure 26: Comparison of the standard off-policy reweighting scheme and the transition reweighting (TO) proposed in this paper.

speed, is similar to other methods if LSTD and LSPE are used with transition reweighting. LSTD and LSPE with transition reweighting obtain the most accurate estimates of all methods as Table 7 indicates.

Message 9 *For a modest number of features, least-squares methods are superior to gradient-based approaches both in terms of data-efficiency and even CPU-time if we want to reach the same error level. For a very large number of features (e.g., $\geq$ 20,000), gradient-based methods should be preferred as least-squares approaches become prohibitively time- and memory-consuming.*

Except for the artifact in the linearized cart-pole balancing task with imperfect features (Benchmarks 5 and 6), least-squares methods yield the most accurate final predictions (see Table 7). However, least-squares approaches may behave very unstable for a small number of

observations. A strong regularization is usually required if the number of samples is smaller than the number of features. An example is given in Figure 17 that shows the increase of the error of BRM predictions for the first 500 samples. However, Figures 22a, 21, 17 and 25 show that least-squares approaches converge much faster than gradient-based methods after this stage of potential instability. Least-squares methods are therefore more data efficient than gradient-based methods.

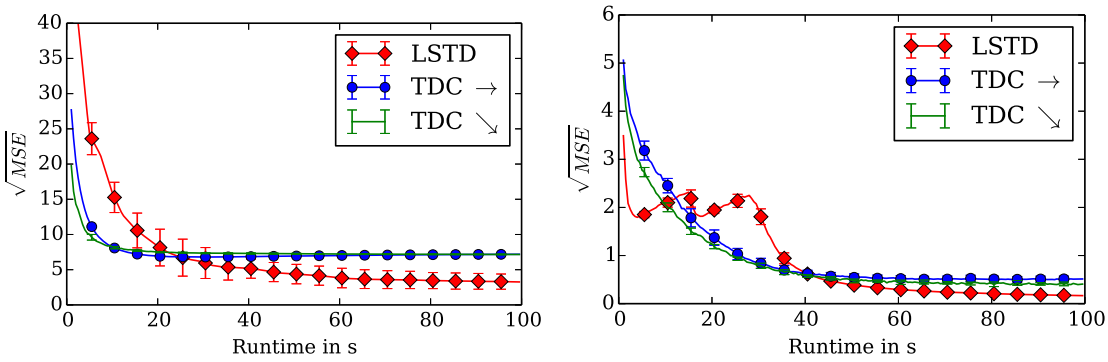
However, the required CPU-time per transition is quadratic in the number of features instead of linear as for the gradient approaches. Hence, it is also interesting to compare both approaches from a computational viewpoint with a fixed budget of CPU-time. Figure 27 compares the prediction quality of LSTD and TDC, the best-performing representatives of both classes, for a given budget of CPU-time. In order to compare the performance on a task with a vast number of features, we changed the number of dimensions in the pendulum balancing task from 20 to 100 and used the perfect feature representation of 20101 dimensions.[10] LSTD requires more CPU time to converge since TDC can do several sweeps through the provided transition samples (7000 in total) while LSTD can update the parameters only a few times due to the high-dimensional features. Yet, TDC converges faster only up to an error level of approximately 8, both for constant and decreasing step-sizes. The prediction error of TDC with decreasing step-sizes still decreases, and will eventually reach the same minimum as the one of LSTD, but very slowly. This observation is consistent with results on stochastic gradient methods in general (see Sra et al., 2012, Chapter 4.1.2). Additionally, we evaluated the methods on a 30-link pendulum with a moderate number of 1830 features. The results are shown in Figure 27b. Due to the smaller number of features LSTD converges faster from the beginning on. However, as the stagnant prediction error up to second 30 shows, LSTD may still yield unstable results as it has not processed enough observations to ensure that its $\boldsymbol{A}_t$-matrix (cf. Equation 25) is invertible and needs strong regularization.

Least-squares methods clearly outperform gradient-based approaches for problems with few features. The quadratic run-time and memory consumption as well as the unstable behavior with few observations become more problematic for increasing numbers of features but, as our results show, least-squares methods may still be a good option for up to 20.000 features

3.4.1 Alternative Regularization Approaches

We implemented and evaluated alternative regularization approaches for LSTD in preliminary experiments, including LARS-TD, LSTD with ℓ_2, ℓ_1, LSTD-ℓ_1 and D-LSTD. However, we observed no performance gain for our benchmarks in comparison to ℓ_2-regularization. We attribute this result to the fact that most of the features in our benchmarks had sufficient quality. The sparse solutions produced by alternative regularization schemes had thus no significant advantage as the noise introduced by active low-quality features in non-sparse solutions was not large enough. We also did not observe the theoretically derived benefits of LSTD with random projections (LSTD-RP), which only become important for extremely many features.

10. The features include the products of all state variables, cf. the features of Benchmark 7.



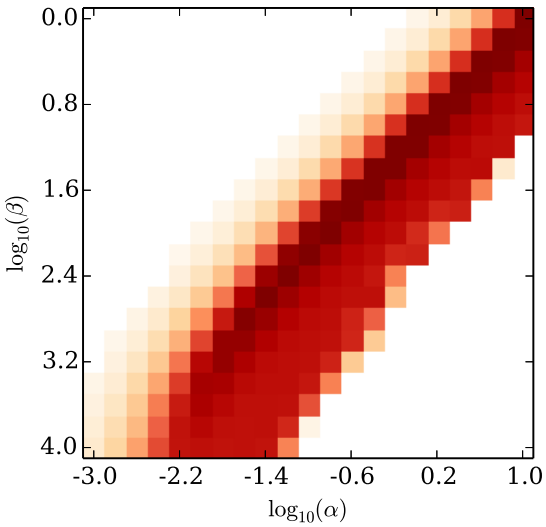
(a) 100-link Pole Balancing with 20101 features (400 dim. state, 20100 squared state features + 1 constant dim.)

(b) 30-link Pole Balancing with 1831 features (60 dim. state, 1830 squared state features + 1 constant dim.)

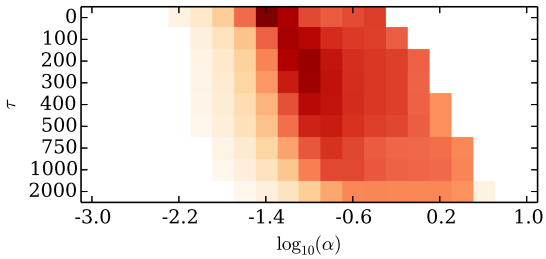
Figure 27: Comparison of the prediction quality for given CPU times of LSTD and TDC with constant (TDC →) and decreasing step-sizes (TDC ↘). The methods are evaluated on multi-link Pole Balancing tasks (in analogy to Benchmark 11) with perfect feature representations. The methods are provided with a total of 7000 transitions. The results are averages of 10 independent runs executed on a single core of an i7 Intel CPU.

3.4.2 Dependency on Hyper-Parameters

Most least-squares methods are robust against a poor choice of the hyper-parameters. LSTD and BRM, in particular, which are controlled only by the regularization parameter ϵ and the parameter λ of the eligibility-traces, converge for almost all values (cf. Figure 18). In contrast, FPKF has four hyper-parameters to optimize, the eligibility-trace parameter λ; a general scaling factor α_{FPKF} for the step-sizes; β_{FPKF} delaying the decrease of the step-length and τ_{FPKF} which controls the minimum number of observations necessary to update the estimate. In particular, α_{FPKF} and β_{FPKF} need to be set correctly to prevent FPKF from diverging as the large white areas in Figure 28a indicate. Also, τ_{FPKF} has large influence on the performance and may cause divergence (Figure 28b). Hence, descent-based optimization strategies such as block gradient descent (with finite differences) are difficult to use for hyper-parameter search as choosing an initial hyper-parameter setting that works is not trivial. On the contrary, LSPE does not rely as much on well-chosen hyper-parameters. As LSTD and BRM, it has an eligibility-traces parameter λ and a regularization-intensity ϵ, but also incorporates a step-size α, which affects the prediction in a similar way as ϵ, that is, it controls the amount of learning. Representative results, given in Figure 29, illustrate that LSPE performs well and stable up to a certain step-size but then diverges. Still, LSTD does not require any step-size at all and performs similarly or better than LSPE (see also Message 7).



(a) Step-size scale α_{FPKF} and length-decreasing parameter β_{FPKF}



(b) Step-size scale α_{FPKF} and the initial update delay τ_{FPKF}

Figure 28: Hyper-parameter space of FPKF(λ) for Benchmark 5. Each point is the logarithm of the averaged MSE. Darker colors show lower errors, while white indicates divergence. The color-maps are log-normalized. Each plot shows a slice of the 4-dimensional hyper-parameter space around the setting used in our experiments (optimal w.r.t. the MSPBE).

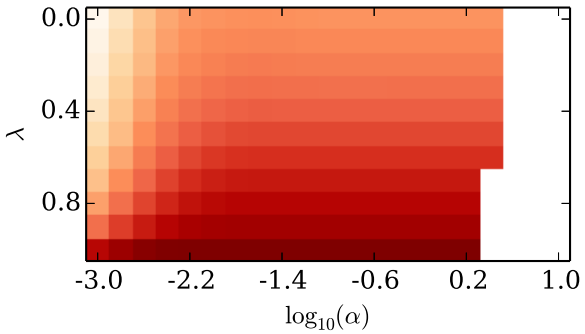


Figure 29: Hyper-parameter space of LSPE(λ) for Benchmark 5. Each point is the logarithm of the averaged MSE. Darker colors show lower errors, while white indicates divergence. The color-maps are log-normalized. The step-size α has little influence on the performance, while e-traces may speed-up learning significantly.

4. Conclusion and Outlook

We conclude the paper with a short summary of its main contributions and a brief outlook on possible directions for future research on temporal-difference methods.

4.1 Conclusion

With this paper, we aimed at giving an exhaustive survey of past and current research activities on value-function estimation with temporal differences—both from a theoretical and an empirical point of view. Almost all important methods originated in this area of research have been presented from a unifying viewpoint of function optimization. The algorithms have been systematically categorized based on their underlying cost functions and the employed optimization technique: stochastic gradient descent, analytic least-squares solutions or a probabilistic problem formulation. We aimed for a concise, yet comprehensive and coherent presentation to make these algorithms available to novices and practitioners.

In addition, we provide an overview over recent work on feature representations for value function approximation. These developments aim either at an automatic generation of state features or at more robust methods that can deal with very large numbers of irrelevant features. We also have provided a qualitative analysis of conceptual error sources that aids novices in understanding the effects of eligibility traces which implicitly perform multi-step look-ahead. Discerning the sources of errors is important for choosing the most suitable estimation method given a new task at hand and for identifying reasons why a particular approach might not work. Furthermore, we have discussed the use of importance reweighting for implementing off-policy value estimation. We have shown that the commonly employed importance reweighting strategy of least-squares methods such as LSTD and LSPE is unsuitable for non-trivial tasks due to its high variance. To alleviate this problem, we have introduced a novel importance reweighting strategy with drastically reduced variance. Our importance reweighting strategy works well in practice—even where the standard reweighting strategy exhibits strong instabilities and yields unsuitable results.

One of the most important contribution of this paper is that it is one of the first comprehensive experimental comparisons of the different value-function estimation methods, including the recent developments. Their performance was evaluated on 12 different benchmark tasks that exhibited different characteristics, including MDPs with continuous and discrete state spaces as well as on- and off-policy transition samples. Our work also provides further evidence on relevant future research questions, such as, which objective function has the lowest bias in practice or which algorithms are preferable in terms of data efficiency or computational demands. Moreover, the experimental evaluation reveals the behavior and the limitations of each temporal-difference method in various scenarios. These findings will hopefully give insights for improving the state of the art in policy evaluation.

4.2 Outlook

Efficient estimation of value functions is a corner stone of reinforcement learning as the value function is needed in the policy improvement step of the many reinforcement learning algorithms. Temporal-difference methods have been used since the late 1980s to estimate the value function of a policy and have since then been an active field of research. In recent

years, the research concentrated on overcoming the instability of the TD learning method with off-policy samples, improving the sample efficiency, or value-function estimation in high-dimensional feature spaces. This research resulted in the development of the current state of the art, such as LSTD or TD learning with Gradient Correction. In addition, the theoretical analysis from different view-points has led to a good theoretical understanding of the foundations of temporal-difference methods.

However, the use of temporal-difference methods has been restricted by several limitations of the methods. In many domains, the assumption of having a compact and informative feature representation is not realistic. Such features are often difficult to define by hand, for example, in real world robotic systems, in health care diagnosis or for controlling of prostheses. We therefore expect the recent efforts of learning the feature representation (cf. Section 2.3) to continue and increase substantially.

Additionally, the number of samples necessary for learning value functions is still prohibitively large for many real-world scenarios, especially those that involve hardware such as robots or other complex, expensive equipment. One way for addressing this shortcoming is to make the learning problem easier by incorporating prior knowledge. Domain knowledge can usually be incorporated most easily in model-based methods that learn the underlying forward dynamics of the task. Such models can be used to create samples in simulation without hardware or by directly using dynamic programming with the learned model. Deisenroth and Rasmussen (2011) successfully learned complex robot control policies by simultaneously learning a system model and using this model to optimize the policy (direct model-based policy search). As their work show, it is crucial to include the uncertainty about the learned model into the actual estimation problem. Value-function estimation methods that can incorporate a learned model and its uncertainty efficiently are therefore a promising direction for future research.

In this paper, we have treated minimizing the mean squared error of a value function estimate as the ultimate goal. However, in most cases the value function is only an intermediate result used for improving the policy. What counts in the end is not the quality of the value function approximation but the quality of the policy after the policy improvement step. Other objective functions might exist that are easier to minimize and do not harm the convergence properties of policy iteration schemes. Characterizing such objectives can prospectively lead to algorithms with faster convergence to an optimal or at least viable policy.

This survey and comparison solely concentrated on policy evaluation. A comprehensive survey and experimental evaluation of temporal differences in policy iteration which builds on the results of this paper is left as future work and strongly needed by the reinforcement learning community.

Acknowledgments

The research leading to these results has received funding from the European Community's Framework Programme CompLACS (FP7-ICT-2009-6 Grant.No.270327).

Appendix A. Derivation of Least-Squares Temporal-Difference Learning

The derivation of the LSTD algorithm begins with rewriting the MSPBE from Equation (9) in terms of a different norm. While this formulation is, strictly speaking, not necessary, it helps to understand the connection to the TDC, GTD and GTD2 algorithms and let us derive subsequent steps more concisely

$$\begin{aligned}
\text{MSPBE}(\boldsymbol{\theta}) &= \|\boldsymbol{V}_{\boldsymbol{\theta}} - \Pi T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}}\|_{\boldsymbol{D}}^2 \\
&= \|\Pi(\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}})\|_{\boldsymbol{D}}^2 \quad \text{(since } \boldsymbol{V}_{\theta} \text{ is parametrizable)} \\
&= [\Pi(\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}})]^T \boldsymbol{D} [\Pi(\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}})] \\
&= [\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}}]^T \Pi^T \boldsymbol{D} \Pi [\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}}] \\
&= [\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}}]^T \boldsymbol{D} \boldsymbol{\Phi} (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi} (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} [\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}}] \\
&= [\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}}]^T \boldsymbol{D} \boldsymbol{\Phi} (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} [\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}}] \\
&= [\boldsymbol{\Phi}^T \boldsymbol{D} (\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}})]^T (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1} [\boldsymbol{\Phi}^T \boldsymbol{D} (\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}})] \\
&= \|\boldsymbol{\Phi}^T \boldsymbol{D} (\boldsymbol{V}_{\boldsymbol{\theta}} - T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}})\|_{(\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1}}^2 \qquad (41) \\
&= \|\mathbb{E}_{d,\pi,\mathcal{P}}[\boldsymbol{\phi}_t \delta_t]\|_{(\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1}}^2 . \qquad (42)
\end{aligned}$$

The matrix $\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi}$ and its inverse $\boldsymbol{M} = (\boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{\Phi})^{-1}$ are symmetric positive definite matrices (for independent features and $d(s) > 0, \forall s \in \mathcal{S}$). Hence, $\| \cdot \|_{\boldsymbol{M}}$ is a norm and $\boldsymbol{\theta}$ minimizes the MSPBE if and only if $\mathbb{E}_{d,\pi,\mathcal{P}}[\boldsymbol{\phi}_t \delta_t] = 0$. Equation (42) also allows us to rewrite the MSPBE as a product of expectations

$$\text{MSPBE}(\boldsymbol{\theta}) = \mathbb{E}_{d,\pi,\mathcal{P}}[\boldsymbol{\phi}_t \delta_t]^T \mathbb{E}_d[\boldsymbol{\phi}_t \boldsymbol{\phi}_t^T]^{-1} \mathbb{E}_{d,\pi,\mathcal{P}}[\boldsymbol{\phi}_t \delta_t], \qquad (43)$$

which is the basis for the GTD2 and TDC algorithms. Since $V_{\boldsymbol{\theta}}$ is parameterized linearly, we can replace $T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}}$ with

$$T^{\pi} \boldsymbol{V}_{\boldsymbol{\theta}} = \boldsymbol{R} + \gamma \boldsymbol{\Phi}' \boldsymbol{\theta},$$

where $\boldsymbol{R} \in \mathbb{R}^n$ is the expected intermediate reward $\boldsymbol{R}_i = \mathbb{E}_{\pi}[r(s^i, a)]$ in state s^i and $\boldsymbol{\Phi}' = \boldsymbol{P}^{\pi} \boldsymbol{\Phi}$ is the matrix containing the expected feature for the successor states, that is, $\boldsymbol{\Phi}'_i = \mathbb{E}_{\mathcal{P},\pi}[\boldsymbol{\phi}_{t+1}^T | s_t = s^i]$. Equation (41) can then be written as

$$\begin{aligned}
\text{MSPBE}(\boldsymbol{\theta}) &= \|\boldsymbol{\Phi}^T \boldsymbol{D} (\boldsymbol{\Phi} \boldsymbol{\theta} - \gamma \boldsymbol{\Phi}' \boldsymbol{\theta} - \boldsymbol{R})\|_{\boldsymbol{M}}^2 \\
&= \|\boldsymbol{\Phi}^T \boldsymbol{D} [\boldsymbol{\Phi} - \gamma \boldsymbol{\Phi}'] \boldsymbol{\theta} - \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{R})\|_{\boldsymbol{M}}^2 \\
&= \|\boldsymbol{\Phi}^T \boldsymbol{D} \Delta \boldsymbol{\Phi} \boldsymbol{\theta} - \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{R})\|_{\boldsymbol{M}}^2 \\
&= \|\boldsymbol{A} \boldsymbol{\theta} - \boldsymbol{b}\|_{\boldsymbol{M}}^2,
\end{aligned}$$

where $\Delta \boldsymbol{\Phi} = \boldsymbol{\Phi} - \gamma \boldsymbol{\Phi}'$ and $\boldsymbol{b} = \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{R}$. The matrix $\boldsymbol{A} = \boldsymbol{\Phi}^T \boldsymbol{D} \Delta \boldsymbol{\Phi}$ has been shown to be positive definite and thus invertible (Bertsekas and Tsitsiklis, 1996, Proposition 6.3.3). Minimizing this MSPBE formulation directly by setting the gradient to 0 yields

$$\boldsymbol{\theta} = (\boldsymbol{A}^T \boldsymbol{M} \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{M} \boldsymbol{b} = \boldsymbol{A}^{-1} \boldsymbol{M}^{-1} \boldsymbol{A}^{-T} \boldsymbol{A}^T \boldsymbol{M} \boldsymbol{b} = \boldsymbol{A}^{-1} \boldsymbol{b} = (\boldsymbol{\Phi}^T \boldsymbol{D} \Delta \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D} \boldsymbol{R}.$$

Appendix B. Parametric GPTD Whitening Transformation

Equation (32) can also be written in matrix form in terms of the reward, that is,

$$\boldsymbol{r}_{t-1} = \boldsymbol{\Gamma}_t \boldsymbol{V}_t + \boldsymbol{n}_t, \qquad \boldsymbol{\Gamma}_t = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & \dots & 0 & 1 & -\gamma \end{bmatrix},$$

where $\boldsymbol{\Gamma}_t$ connects the values of subsequent timesteps, $\boldsymbol{r}_{t-1} = [r_1, \dots, r_{t-1}]$ and $\boldsymbol{V}_t = [v_1 \dots v_t]$. The noise term $\boldsymbol{n}_t$ is now given as $\boldsymbol{n}_t = \boldsymbol{\Gamma}_t \Delta \boldsymbol{v}_t$, and hence is distributed as $\boldsymbol{n}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_t)$, with

$$\boldsymbol{\Sigma}_t = \sigma^2 \boldsymbol{\Gamma}_t \boldsymbol{\Gamma}_t^T = \sigma^2 \begin{bmatrix} 1+\gamma^2 & -\gamma & 0 & \dots & 0 \\ -\gamma & 1+\gamma^2 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & -\gamma & 1+\gamma^2 \end{bmatrix}.$$

We realize that the required whitening transformation is given by

$$\boldsymbol{Z}_t = \boldsymbol{\Gamma}_t^{-1} = \begin{bmatrix} 1 & \gamma & \gamma^2 & \dots & \gamma^t \\ 0 & 1 & \gamma & \dots & \gamma^{t-1} \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix},$$

and hence, we get the following regression problem

$$\boldsymbol{Z}_t \boldsymbol{r}_{t-1} = \boldsymbol{Z}_t \boldsymbol{\Gamma}_t \boldsymbol{V}_t + \boldsymbol{Z}_t \boldsymbol{n}_t.$$

Appendix C. Algorithms

The following pseudo-code listings show the update rules of all discussed 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.

Algorithm 1 TD(λ) Learning

$$\begin{aligned} \boldsymbol{z}_{t+1} &= \rho_t(\boldsymbol{\phi}_t + \lambda\gamma \boldsymbol{z}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t \delta_t \boldsymbol{z}_{t+1} \end{aligned}$$

Algorithm 2 GTD

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t \rho_t (\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}) \boldsymbol{\phi}_t^T \boldsymbol{w}_t \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \beta_t \rho_t (\delta_t \boldsymbol{\phi}_t - \boldsymbol{w}_t) \end{aligned}$$

Algorithm 3 GTD2

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t \rho_t (\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}) \boldsymbol{\phi}_t^T \boldsymbol{w}_t \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \beta_t (\rho_t \delta_t - \boldsymbol{\phi}_t^T \boldsymbol{w}_t) \boldsymbol{\phi}_t \end{aligned}$$

Algorithm 4 TDC(λ)

$$\begin{aligned} \boldsymbol{z}_{t+1} &= \rho_t(\boldsymbol{\phi}_t + \lambda\gamma \boldsymbol{z}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t (\delta_t \boldsymbol{z}_t - \gamma(1-\lambda)(\boldsymbol{z}_t^T \boldsymbol{w}_t) \boldsymbol{\phi}_{t+1}) \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \beta_t (\rho_t \delta_t \boldsymbol{z}_t - \boldsymbol{\phi}_t^T \boldsymbol{w}_t \boldsymbol{\phi}_t) \end{aligned}$$

Algorithm 5 recursive LSTD(λ) (Init: $\boldsymbol{M}_0 = \epsilon \boldsymbol{I}$)

$$
\begin{aligned}
\Delta\boldsymbol{\phi}_{t+1} =& \boldsymbol{\phi}_t - \rho_t \gamma \boldsymbol{\phi}_{t+1} \\
\boldsymbol{z}_t =& \gamma\lambda\rho_{t-1}\boldsymbol{z}_{t-1} + \boldsymbol{\phi}_t \\
\boldsymbol{K}_{t+1} =& \frac{\boldsymbol{M}_t \boldsymbol{z}_t}{1 + \Delta\boldsymbol{\phi}_{t+1}^T \boldsymbol{M}_t \boldsymbol{z}_t} \\
\boldsymbol{\theta}_{t+1} =& \boldsymbol{\theta}_t + \boldsymbol{K}_{t+1}(\rho_t r_t - \Delta\boldsymbol{\phi}_{t+1})^T \boldsymbol{\theta}_t) \\
\boldsymbol{M}_{t+1} =& \boldsymbol{M}_t - \boldsymbol{K}_{t+1}(\boldsymbol{M}_t^T \Delta\boldsymbol{\phi}_{t+1})^T
\end{aligned}
$$

Algorithm 6 recursive LSTD-TO(λ) (Init: $\boldsymbol{M}_0 = \epsilon \boldsymbol{I}$)

$$
\begin{aligned}
\Delta\boldsymbol{\phi}_{t+1} =& \boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1} \\
\boldsymbol{z}_t =& \gamma\lambda\rho_{t-1}\boldsymbol{z}_{t-1} + \boldsymbol{\phi}_t \\
\boldsymbol{K}_{t+1} =& \rho_t \frac{\boldsymbol{M}_t \boldsymbol{z}_t}{1 + \rho_t \Delta\boldsymbol{\phi}_{t+1}^T \boldsymbol{M}_t \boldsymbol{z}_t} \\
\boldsymbol{\theta}_{t+1} =& \boldsymbol{\theta}_t + \boldsymbol{K}_{t+1}(r_t - \Delta\boldsymbol{\phi}_{t+1}^T \boldsymbol{\theta}_t) \\
\boldsymbol{M}_{t+1} =& \boldsymbol{M}_t - \boldsymbol{K}_{t+1}(\boldsymbol{M}_t^T \Delta\boldsymbol{\phi}_{t+1})^T
\end{aligned}
$$

Algorithm 7 recursive LSPE(λ) (Init: $\boldsymbol{N}_0 = \epsilon \boldsymbol{I}, \boldsymbol{A}_0 = \boldsymbol{0}, \boldsymbol{b}_0 = \boldsymbol{0}$)

$$
\begin{aligned}
\boldsymbol{z}_t =& \gamma\lambda\rho_{t-1}\boldsymbol{z}_{t-1} + \boldsymbol{\phi}_t \\
\boldsymbol{N}_{t+1} =& \boldsymbol{N}_t - \frac{\boldsymbol{N}_t \boldsymbol{\phi}_t \boldsymbol{\phi}_t^T \boldsymbol{N}_t}{1 + (\boldsymbol{\phi}_t^T \boldsymbol{N}_t \boldsymbol{\phi}_t)} \\
\boldsymbol{A}_{t+1} =& \boldsymbol{A}_t + \boldsymbol{z}_t(\boldsymbol{\phi}_t - \gamma\rho_t \boldsymbol{\phi}_{t+1}))^T \\
\boldsymbol{b}_{t+1} =& \boldsymbol{b}_t + \rho_t r_t \boldsymbol{z}_t \\
\boldsymbol{\theta}_{t+1} =& \boldsymbol{\theta}_t + \alpha_t \boldsymbol{N}_t(\boldsymbol{b}_t - \boldsymbol{A}_t \boldsymbol{\theta}_t)
\end{aligned}
$$

Algorithm 8 recursive LSPE-TO(λ) (Init: $\boldsymbol{N}_0 = \epsilon \boldsymbol{I}, \boldsymbol{A}_0 = \boldsymbol{0}, \boldsymbol{b}_0 = \boldsymbol{0}$)

$$
\begin{aligned}
\boldsymbol{z}_t =& \gamma\lambda\rho_{t-1}\boldsymbol{z}_{t-1} + \boldsymbol{\phi}_t \\
\boldsymbol{N}_{t+1} =& \boldsymbol{N}_t - \frac{\boldsymbol{N}_t \boldsymbol{\phi}_t \boldsymbol{\phi}_t^T \boldsymbol{N}_t}{1 + (\boldsymbol{\phi}_t^T \boldsymbol{N}_t \boldsymbol{\phi}_t)} \\
\boldsymbol{A}_{t+1} =& \boldsymbol{A}_t + \rho_t \boldsymbol{z}_t(\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}))^T \\
\boldsymbol{b}_{t+1} =& \boldsymbol{b}_t + \rho_t r_t \boldsymbol{z}_t \\
\boldsymbol{\theta}_{t+1} =& \boldsymbol{\theta}_t + \alpha_t \boldsymbol{N}_t(\boldsymbol{b}_t - \boldsymbol{A}_t \boldsymbol{\theta}_t)
\end{aligned}
$$

Algorithm 9 FPKF(λ) (Init: $\boldsymbol{N}_0 = \boldsymbol{0}$, $\boldsymbol{Z}_0 = \boldsymbol{0}, \boldsymbol{z}_0 = \boldsymbol{0}$)

$$
\begin{aligned}
\boldsymbol{z}_t =& \gamma\lambda\rho_{t-1}\boldsymbol{z}_{t-1} + \boldsymbol{\phi}_t \\
\boldsymbol{Z}_t =& \gamma\lambda\rho_{t-1}\boldsymbol{Z}_{t-1} + \boldsymbol{\phi}_t \boldsymbol{\theta}_t^T \\
\boldsymbol{N}_{t+1} =& \boldsymbol{N}_t - \frac{\boldsymbol{N}_t \boldsymbol{\phi}_t \boldsymbol{\phi}_t^T \boldsymbol{N}_t}{1 + (\boldsymbol{\phi}_t^T \boldsymbol{N}_t \boldsymbol{\phi}_t)} \\
\boldsymbol{\theta}_{t+1} =& \boldsymbol{\theta}_t + \alpha_t \boldsymbol{N}_t(\boldsymbol{z}_t \rho_t r_t - \boldsymbol{Z}_t(\boldsymbol{\phi}_t - \gamma\rho_t \boldsymbol{\phi}_{t+1}))
\end{aligned}
$$

Algorithm 10 recursive BRM with double samples (Init: $\boldsymbol{M}_0 = \epsilon \boldsymbol{I}$, $\boldsymbol{b}_0 = \boldsymbol{0}$)

$$
\begin{aligned}
\Delta\boldsymbol{\phi}'_{t+1} =& \boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}'_{t+1} \\
\Delta\boldsymbol{\phi}''_{t+1} =& \boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}''_{t+1} \\
\boldsymbol{b}_{t+1} =& \boldsymbol{b}_t + \frac{1}{2}\rho'_t \rho''_t (\Delta\boldsymbol{\phi}''_{t+1} r'_t + \Delta\boldsymbol{\phi}'_{t+1} r''_t) \\
\boldsymbol{M}_{t+1} =& \boldsymbol{M}_t - \frac{\rho'_t \rho''_t \boldsymbol{M}_t \Delta\boldsymbol{\phi}'_{t+1} \Delta\boldsymbol{\phi}''^T_{t+1} \boldsymbol{M}_t}{1 + \rho'_t \rho''_t \Delta\boldsymbol{\phi}'^T_{t+1} \boldsymbol{M}_t \Delta\boldsymbol{\phi}''_{t+1}} \\
\boldsymbol{\theta}_{t+1} =& \boldsymbol{M}_{t+1} \boldsymbol{b}_{t+1}
\end{aligned}
$$

Algorithm 11 recursive BRM(λ)
(Init: $\boldsymbol{M}_0 = \epsilon \boldsymbol{I}, \boldsymbol{x}_0 = 0, y_0 = 1, z_0 = 0$)

Compute auxiliary values:

$$\begin{aligned}
\Delta\boldsymbol{\phi}_{t+1} =& \boldsymbol{\phi}_t - \gamma\rho_t\boldsymbol{\phi}_{t+1} \\
p_{t+1} =& \frac{\gamma\lambda\rho_{t-1}}{\sqrt{y_t}} \\
\boldsymbol{U}_{t+1} =& \left[\sqrt{y_t}\Delta\boldsymbol{\phi}_{t+1} + p_{t+1}\boldsymbol{x}_t, \quad p_{t+1}\boldsymbol{x}_t\right] \\
\boldsymbol{V}_{t+1} =& \left[\sqrt{y_t}\Delta\boldsymbol{\phi}_{t+1} + p_{t+1}\boldsymbol{x}_t, \ -p_{t+1}\boldsymbol{x}_t\right]^T \\
\boldsymbol{W}_{t+1} =& \left[\sqrt{y_t}\rho_t r_t + p_{t+1}z_t, \qquad -p_{t+1}z_t\right]^T \\
\boldsymbol{B}_{t+1} =& \boldsymbol{M}_t\boldsymbol{U}_{t+1}[\boldsymbol{I} + \boldsymbol{V}_{t+1}\boldsymbol{M}_t\boldsymbol{U}_{t+1}]^{-1}
\end{aligned}$$

Update traces:

$$\begin{aligned}
\boldsymbol{M}_{t+1} =& \boldsymbol{M}_t - \boldsymbol{B}_{t+1}\boldsymbol{V}_{t+1}\boldsymbol{M}_t \\
y_{t+1} =& (\gamma\lambda\rho_t)^2 y_t + 1 \\
\boldsymbol{x}_{t+1} =& (\gamma\lambda\rho_{t-1})\boldsymbol{x}_t + y_t\Delta\phi_{t+1} \\
z_{t+1} =& (\gamma\lambda\rho_{t-1})z_t + r_t\rho_t y_t
\end{aligned}$$

Update estimate:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{B}_{t+1}(\boldsymbol{W}_{t+1} - \boldsymbol{V}_{t+1}\boldsymbol{\theta}_t)$$

Algorithm 12 parametric GPTD (Init: $\boldsymbol{P}_0 = \boldsymbol{I}$, $\boldsymbol{p}_o = \boldsymbol{0}, d_0 = 0$, $s_0^{-1} = 0$)

$$\begin{aligned}
\Delta\boldsymbol{\phi}_{t+1} =& \boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1} \\
\boldsymbol{p}_{t+1} =& \boldsymbol{p}_t\frac{\gamma\sigma_t^2}{s_t} + \boldsymbol{P}_t\Delta\boldsymbol{\phi}_{t+1} \\
d_{t+1} =& d_t\frac{\gamma\sigma_t^2}{s_t} + r_t - \Delta\boldsymbol{\phi}_{t+1}^T\boldsymbol{\theta}_t \\
s_{t+1} =& \sigma_t^2 + \gamma^2\sigma_{t+1}^2 - \frac{\gamma^2\sigma_t^4}{s_t} \\
& + \left[\boldsymbol{p}_{t+1} + \frac{\gamma\sigma_t^2}{s_t}\boldsymbol{p}_t\right]^T\Delta\boldsymbol{\phi}_{t+1} \\
\boldsymbol{\theta}_{t+1} =& \boldsymbol{\theta}_t + \frac{1}{s_{t+1}}\boldsymbol{p}_{t+1}d_{t+1} \\
\boldsymbol{P}_{t+1} =& \boldsymbol{P}_t - \frac{1}{s_{t+1}}\boldsymbol{p}_{t+1}\boldsymbol{p}_{t+1}^T
\end{aligned}$$

Algorithm 13 Residual-gradient algorithm without double-samples

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t\rho_t\delta_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})$$

References

J. S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems Measurement and Control*, 97(September): 220–227, 1975.

S.-i. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2): 251–276, Feb. 1998.

A. Antos, C. Szepesvári, and R. Munos. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

F. Bach and E. Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems 24*, 2011.

L. Baird. Residual algorithms : Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

P. Balakrishna, R. Ganesan, and L. Sherry. 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, 2010.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996. ISBN 1-886529-10-8.

D. P. Bertsekas and H. Yu. Projected equation methods for approximate solution of large linear systems. *Journal of Computational and Applied Mathematics*, 227(1):27–50, 2009.

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

S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

E. Candes and T. Tao. The Dantzig selector: statistical estimation when p is much larger than n. *The Annals of Statistics*, 35(6):2313–2351, 2005.

D. Choi and B. Roy. A generalized Kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynamic Systems*, 16(2):207–239, 2006.

R. W. Cottle, J.-S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Computer Science and Scientific Computing. Academic Press, 1992. ISBN 0121923509.

R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2-3):235–262, 1998.

W. Dabney and A. G. Barto. Adaptive step-size for online temporal difference learning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, (1):19–67, 2010.

M. P. Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes*. PhD thesis, Karlsruhe Institute of Technology, 2010.

M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.

Y. Engel. *Algorithms and Representations for Reinforcement Learning*. PhD thesis, Hebrew University, 2005.

Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

A.-m. Farahmand and C. Szepesvári. Model selection in reinforcement learning. *Machine Learning*, 85(3):299–332, 2011.

A.-m. Farahmand, M. Ghavamzadeh, C. Szepesvári, and S. Mannor. Regularized policy iteration. In *Advances in Neural Information Processing Systems 21*, 2008.

J. Frank, S. Mannor, and D. Precup. Reinforcement learning in the presence of rare events. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

M. Geist and O. Pietquin. Kalman temporal differences. *Journal of Artificial Intelligence Research*, 39(1):483–532, 2010.

M. Geist and B. Scherrer. l1-penalized projected Bellman residual. In *Proceedings of the Nineth European Workshop on Reinforcement Learning*, 2011.

M. Geist and B. Scherrer. Off-policy learning with eligibility traces : A survey. Technical report, INRIA Lorraine - LORIA, 2013.

M. Geist, B. Scherrer, A. Lazaric, and M. Ghavamzadeh. A Dantzig selector approach to temporal difference learning. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

S. Gelly and D. Silver. Achieving master level play in 9 x 9 computer go. In *Proceedings of the 23th AAAI Conference on Artificial Intelligence*, 2008.

A. Geramifard, M. Bowling, and R. S. Sutton. Incremental least-squares temporal difference learning. *Proceedings of the 21th AAAI Conference on Artificial Intelligence*, 2006a.

A. Geramifard, M. Bowling, M. Zinkevich, and R. S. Sutton. iLSTD: Eligibility traces and convergence analysis. In *Advances in Neural Information Processing Systems 19*, 2006b.

A. Geramifard, F. Doshi, J. Redding, N. Roy, and J. P. How. Online discovery of feature dependencies. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

A. Geramifard, T. J. Walsh, and J. P. How. Batch-iFDD for representation expansion in large MDPs. In *Conference on Uncertainty in Artificial Intelligence*, 2013.

M. Ghavamzadeh, A. Lazaric, O.-A. Maillard, and R. Munos. LSTD with random projections. In *Advances in Neural Information Processing Systems 23*, 2010.

M. Ghavamzadeh, A. Lazaric, R. Munos, and M. Hoffman. Finite-sample analysis of Lasso-TD. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

P. W. Glynn and D. L. Iglehart. Importance sampling for stochastic simulations. *Management Science*, 35(11):1367–1392, 1989.

H. Hachiya and M. Sugiyama. Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2010.

M. Hoffman, A. Lazaric, M. Ghavamzadeh, and R. Munos. Regularized least squares temporal difference learning with nested l2 and l1 penalization. In *Proceedings of the Nineth European Workshop on Reinforcement Learning*, 2011.

M. Hutter and S. Legg. Temporal difference updating without a learning rate. In *Advances in Neural Information Processing Systems 20*, 2007.

R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for sparse hierarchical dictionary learning. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

J. Johns and S. Mahadevan. Sparse approximate policy evaluation using graph-based basis functions. Technical report, University of Massachusetts Amherst, 2009.

J. Johns, C. Painter-Wakefield, and R. Parr. Linear complementarity for regularized policy evaluation and improvement. In *Advances in Neural Information Processing Systems 23*, 2010.

T. Jung and D. Polani. Least squares SVM for least squares TD learning. In *European Conference on Artificial Intelligence*, 2006.

P. W. Keller, S. Mannor, and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

J. Z. Kolter and A. Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

R. M. Kretchmar and C. W. Anderson. Comparison of CMACs and radial basis functions for kocal function approximators in reinforcement learning. In *International Conference on Neural Networks*, 1997.

M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4(Dec):1107–1149, 2003.

A. Lazaric, M. Ghavamzadeh, and R. Munos. Finite-sample analysis of LSTD. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

L. Li. A worst-case comparison between temporal difference and residual gradient with linear function approximation. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

B. Liu, S. Mahadevan, and J. Liu. Regularized off-policy TD-learning. In *Advances in Neural Information Processing Systems 25*, 2012.

M. Loth, M. Davy, and P. Preux. Sparse temporal difference learning using LASSO. In *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning*, 2007.

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

S. Mahadevan and M. Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8(Oct):2169–2231, 2007.

A. R. Mahmood, R. S. Sutton, T. Degris, and P. M. Pilarski. Tuning-free step-size adaptation. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012.

I. Menache, S. Mannor, and N. Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.

D. Meyer, H. Shen, and K. Diepold. l1-regularized gradient temporal-difference learning. In *Proceedings of the Tenth European Workshop on Reinforcement Learning*, 2012.

A. Nedic and D. P. Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110, 2003.

C. Painter-Wakefield and R. Parr. Greedy algorithms for sparse reinforcement learning. In *Proceedings of the 29th International Conference on Machine Learning*, 2012a.

C. Painter-Wakefield and R. Parr. L1 regularized linear temporal difference learning. Technical report, Duke University, Durham, NC, 2012b.

R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th International Conference on Machine Learning*, 2007.

R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27th Asilomar Conference on Signals, Systems and Computers*, 1993.

M. Petrik, G. Taylor, R. Parr, and S. Zilberstein. Feature selection using regularization in approximate linear programs for Markov decision processes. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

B. A. Pires. *Statistical Analysis of L1-penalized Linear Estimation with Applications*. Master thesis, University of Alberta, 2011.

C. E. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems 16*, 2003.

M. Riedmiller and T. Gabel. On experiences in a complex and competitive gaming domain: Reinforcement learning meets robocup. In *IEEE Symposium on Computational Intelligence and Games*, 2007.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

M. Rosenblatt. *Markov Processes. Structure and Asymptotic Behavior*. Springer, 1971. ISBN 978-3642652400.

N. L. Roux and A. Fitzgibbon. A fast natural Newton method. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

B. Scherrer. Should one compute the temporal difference fix point or minimize the Bellman residual? the unified oblique projection view. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

B. Scherrer and M. Geist. Recursive least-squares learning with eligibility traces. In *Proceedings of the Nineth European Workshop on Reinforcement Learning*, 2011.

R. Schoknecht. Optimality of reinforcement learning algorithms with linear function approximation. In *Advances in Neural Information Processing Systems 15*, 2002.

P. J. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(2):568–582, 1985.

D. Silver, R. Sutton, and M. Müller. Reinforcement learning of local shape in the game of go. In *International Joint Conference on Artificial Intelligence*, 2007.

S. Sra, S. Nowozin, and S. J. Wright. *Optimization for Machine Learning*. MIT Press, 2012. ISBN 9780262016469.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 9780262193986.

R. S. Sutton, D. Precup, and S. Singh. Intra-option learning about temporally abstract actions. In *Proceedings of the 15th International Conference on Machine Learning*, 1998.

R. S. Sutton, C. Szepesvári, and H. R. Maei. A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In *Advances in Neural Information Processing Systems 21*, 2008.

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

G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

G. Tesauro. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

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

R. J. Williams and L. C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. In *Yale Workshop on Adaptive and Learning Systems*, 1993.

X. Xu, T. Xie, D. Hu, and X. Lu. Kernel least-squares temporal difference learning. *International Journal of Information Technology*, 11(9):54–63, 2005.

H. Yu. Convergence of least squares temporal difference methods under general conditions. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, 37(6A):3468–3497, 2009.