Journal of Machine Learning Research () Submitted ; Published

Gradient-descent Algorithms for Approximate Policy Evaluation by Dynamic Programming and Temporal-difference Learning

Editor:

Abstract

In this article, we summarize and extend recent results with the gradient-TD family of algorithms for approximate policy evaluation in Markov decision processes. In recent years, solution methods based on dynamic programming (DP) and temporal-difference learning (TDL) have been applied to large state spaces using parametric value-function approximation. This approach has had groundbreaking application successes, such as Tesauro's world-champion backgammon program in 1992, but also important failures—examples of divergence are known for both DP and TDL when extended to non-linear approximation or to off-policy learning. We present gradient-descent versions of DP, TD(λ), and Sarsa(λ), called GDP, GTD(λ), and GQ(λ) respectively, that solve these problems. All three algorithms converge for nonlinear function approximators and require computation that is only linear in the number of parameters. The two learning algorithms can work directly from data without a model, simulator, access to the underlying system state, or full control of the policy that generates the data (off-policy learning). We also show how the focus of approximation during off-policy learning can be controlled in a flexible way without compromising the convergence guarantees. Unlike the residual-gradient algorithms developed by Baird in the late 1990s (and more recently extended to Gaussian process reinforcement learning and Kalman TDL) our algorithms converge to a solution of the projected Bellman equation.

Keywords: Temporal-difference learning, Dynamic programming, Reinforcement learning, Value function approximation, Gradient descent, Off-policy, Eligibility traces, Bellman error

1. Issues in value-function approximation

Markov decision processes (MDPs) are a widely used problem formulation in artificial intelligence and throughout engineering. A key idea underlying almost all efficient solution strategies is the computation or estimation of a value function mapping states of the MDP to expected longterm future reward or cost given a decision making policy. For MDPs with large state spaces it is generally necessary to approximate the true value function. Solution methods without approximation, based on dynamic programming (DP) and a tabular representation of the value function, are well understood but suitable only for small problems in which the table can be held in main computer memory. Larger state spaces, even continuous ones, can be handled by DP with discretization, state aggregation, and interpolation,

but again, as state dimensionality increases, these methods rapidly become computationally infeasible or ineffective. This is the effect which gave rise to Bellman's "curse of dimensionality." Large state spaces and the need to effectively and efficiently approximate the value function has long been recognized as an important problem in DP and artificial intelligence. It has been approached in a great many ways by different researchers depending on practical and strategic issues, as well as on their different goals and predilections.

The first strategic issue is whether to approximate the value function for a given policy, termed *policy evaluation*, or to approximate the value function for an optimal policy which is not initially known. We focus on policy evaluation, rather than policy optimization, because it is a simpler problem and yet still includes key open problems. Focusing on policy evaluation allows us to lay aside a host of issues including maintaining sufficient exploration and chattering near optimal policies (see Bertsekas 2012, Chapter 6, for an overview). In policy evaluation there are longstanding open problems regarding robust convergence, including with nonlinear function approximators and off-policy learning, as we detail and partially solve in this paper. To see these hard problems clearly, it is best to address them in the simplest possible setting in which they occur. Another reason for focusing on policy evaluation is that many methods for policy optimization involve evaluation as an intermediate, repeated step; solving policy evaluation better can be expected to lead to better optimization methods. Finally, policy evaluation is an example of longterm prediction, a problem which is of independent interest.

A second issue is whether to use the current value estimates in building improved estimates, as in DP and temporal-difference learning (TDL), or to build solely on the basis of the outcomes of complete trajectories, as in "Monte Carlo" (Sutton & Barto 1998), or "rollout" (Bertsekas 2012), methods. The former, which has been termed "bootstrapping" (Sutton & Barto 1998) is more complex and interesting, and is computationally convenient. In some applications bootstrapping is clearly superior, but in others it is inferior. The theoretical issues are not clear, but intuitively bootstrapping helps to the extent that the state is observable and the function approximator is able to make good value-function estimates. Without bootstrapping, the problem of policy evaluation reduces to conventional supervised learning and needs little specialized treatment; the problems we address in this paper do not arise. In some algorithms, an eligibility trace parameter $\lambda \in [0, 1]$ provides a computationally convenient way of moving smoothly between pure bootstrapping, when $\lambda = 0$, to no bootstrapping, when $\lambda = 1$. In this paper we choose to embrace rather than avoid the complexities of bootstrapping.

The next issue is that of the overall structure of the function approximator. We pursue a general parametric approach in which the value function is represented by an arbitrary smooth approximator of fixed size and structure with many variable parameters, or weights. For example, the approximate value function might be a weighted linear combination of nonlinear features, or a neural network with many layers of neuron-like units and connection weights. The parameters, or weights, are then changed to reshape the approximate value function to better match the true value function. Parametric methods for value function approximation in DP are almost as old as DP itself (e.g., see Bellman & Dreyfus 1959) but became much more popular when DP was combined with sampling methods to produce reinforcement learning and TDL. A key event was Tesauro's (1992, 1995) use of a neural network approximator together with the TD(λ) algorithm (Sutton 1988) to produce

a computer program capable of playing backgammon at a world champion level. The game of backgammon has over 10^{20} states, yet Tesauro's program learned a good approximate value function with less than 50,000 weights. The state was represented to the neural network with a feature vector of 198 components; there does not seem to be any practical way of approximating this value function with tabular, discretizing, or aggregation methods. Since Tesauro's work there have been many similar world-class successes using parametric value-function approximation (e.g., in chess, othello, hearts, tetris, non-game applications?). Non-parametric value-function approximation is also a popular research topic (...).

Finally, in this article, we restrict attention to methods whose computational complexity scales linearly with the number of parameters in the function approximator, like TD(λ), rather than quadratically, like LSTD(λ). This choice deserves some discussion. Least-squares methods like LSTD have many attractive features, but their greater complexity can make them infeasible for large applications. Least-squares methods are generally more efficient in terms of data or iterations before reaching a given level of accuracy. They involve no step-size parameters to be set (though they do have other initialization parameters that may be equally annoying to set manually). On the other hand, it is more difficult to adapt least-squares methods to nonstationary problems, which becomes an issue when moving beyond policy evaluation to policy optimization. However, the biggest drawback to least-squares methods is just the computational one and its consequences. If scaling is linear, then parameters are cheap and one ends up with a lot of them; ten of thousands or millions are used in typical applications such as those mentioned in the previous paragraph. If the application is computation limited, and often it is, then the parameters are reduced to the square root. Instead of 10,000 parameters, one gets 100. Instead of a million, one gets 1000. Instead of being free with parameters, weighting many things just in case any of them is important, one ends up choosing the parameters manually with great care to keep their number down. In a typical case, the parameters are weights on nonlinear features of the state space. Fewer parameters then means fewer features that can be taken into account in the approximate value function. A least-squares method may find the best parameter values faster, but in the long run may have a much worse approximated value function because it can take into account so many fewer features of the state.

The tradeoffs between linear and quadratic complexity in bootstrapping methods such as linear TD(λ) and LSTD(λ) closely parallels similar tradeoffs already seen in supervised learning settings. The classical supervised-learning algorithm for incremental linear regression, known as the LMS (Least Mean Square) algorithm, or Widrow-Hoff rule, is of linear complexity, and the corresponding quadratic complexity algorithm is the least squares (LS) algorithm. LS is more data efficient, even optimal in a sense, but more complex; LMS is often slower to learn but is robust, simpler to implement, and can handle many more parameters and features. In the academic literature, LS is much more popular, but in real-world applications, LMS is more widely used, almost always being prefered over LS according to those with long experience (Widrow, personal communication; Haykin, personal communication). It is also telling that, when neural networks became popular for supervised learning, beginning in the late 1980s and continuing through today, LMS was generalized to backpropagation but quadratic-complexity generalizations have remained little used.

We have experimented with both linear- and quadratic-complexity algorithms for predicting future robot-sensor values in realtime. Making and updating predictions ten times

per second using thousands of features, with linear-complexity methods we were able to predict almost ten thousand different sensory events, whereas with quadratic complexity methods we could predict only one. It is clear to us that there are already cases where computational costs are critical and the advantage of linear methods is decisive. As the power of modern computers increases, we can expect to have more learned parameters and the advantage to linear-complexity methods can be expected only to increase.

Having explained the choices underlying our approach, we can now outline our main results, as summarized in the table in Figure 1. The table has seven columns, two corresponding to DP algorithms and five to TDL algorithms. The first column, for example, corresponds to the classical algorithm TD(λ) (and Sarsa(λ), the analogous algorithm for learning state–action values). The last two rows correspond to the new gradient-TD family of algorithms presented in this article. The rows correspond to five issues or properties that we would like the algorithms to have. First, as discussed just above, we would like the algorithms to have linear computational complexity, and most do, with LSTD(λ) being one of the listed exceptions. Another row corresponds to whether the algorithm will work with general nonlinear function approximators (subject to smoothness conditions, as described below). We see that TD(λ) is linear complexity, but is not guaranteed to converge with nonlinear function approximation. In fact, counterexamples are known. We will show that gradient-TD algorithms converge on any MDP, and in particular on these counterexamples. TD(λ) is also not guaranteed to converge under off-policy training (third row). Again, counterexamples are known, and we show that gradient-TD methods converge on them. Note that according to four of the five properties listed here, TD(λ) and approximate DP have the same properties. (The only difference is that, of course, TD(λ) is a model-free method whereas DP is a model-based method.) Both are linear complexity per update of the function approximator, and both fail with nonlinear approximators and off-policy training. We discuss this in Section 4.

Note that the "Residual gradient" algorithm (Baird 1995, fifth column) does well on all properties except "Converges to PBE=0". All the other algorithms compute or converge to the same asymptotic solution, at which the projected Bellman error (PBE) is zero. The Residual gradient algorithm is the only prior algorithm based on true gradient descent. Gradient descent is a powerful strategy for creating robustly convergent algorithms. Residual gradient algorithms illustrate this strength, but unfortunately the objective function they use does not have its minimum in the right place. Conventional temporal-difference algorithms have a gradient aspect to them, but are known not to be the gradient of any objective function (Barnard 1993). Our gradient-TD methods, on the other hand, are well thought of as gradient-descent methods, converge robustly with both linear function approximation and off-policy training, and converge to zero PBE. All these issues are discussed in Section **??**

So far we have emphasized that parameterized function approximation is a way of handling large state spaces, but it is also a way of handling incompletely observed state. A partially observed MDP (POMDP), in which only observations are available to the agent and not state variables, can always be treated just as a regular MDP with function approximation. A general approach is to map the history of observations and actions to a feature vector and then use a function approximator that takes the feature vector as input

ISSUE	ALGORITHM						
	TD(λ), Sarsa(λ)	Approx. DP	LSTD(λ), LSPE(λ)	Fitted-Q	Residual gradient	GDP	GTD(λ), GQ(λ)
Linear computation	✓	✓	✖	✖	✓	✓	✓
Nonlinear convergent	✖	✖	✖	✓	✓	✓	✓
Off-policy convergent	✖	✖	✓	✖	✓	✓	✓
Model-free, online	✓	✖	✓	✖	✓	✖	✓
Converges to PBE = 0	✓	✓	✓	✓	✖	✓	✓

Figure 1: Issues with bootstrapping algorithms for approximate parametric policy evaluation. There are many aspects of each symbol that deserve further remarks and clarifications, which will go here.

rather than the state. A related issue is the difference between the *planning case*, in which a model of the full MDP is available, and the *learning case*, in which only the data stream is available. Most of what we have to say about function approximation applies to both cases, but there are a few important exceptions, which we will note as they arise.

2. Markov decision processes with value-function approximation

We use the common formulation of a Markov decision process (MDP) as a five-tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ as follows. A decision-making agent interacts with the MDP in discrete time $t = 0, 1, 2, \ldots$. At each time, t, the agent finds itself in state $S_t \in \mathcal{S}$ and selects an action $A_t \in \mathcal{A}$. The MDP then changes state, to state S_{t+1}, with probability $p(S_{t+1}|S_t, A_t)$ (or probability density if $\mathcal{S}$ is continuous) and emits a reward $R_{t+1} \in \mathbb{R}$ according to some probability distribution with expected value $r(s, a)$. The agent may select its actions according to a stationary decision making policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ where $\pi(s, a)$ is the probability that $A_t = a$ given that $S_t = s$, for all t. The object is to maximize the γ-discounted cumulative reward received from each state. To get started, define a random variable G_t, the *return* at time t, as:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots , \tag{1}$$

where the dot above the equals sign indicates that this is a definition, and $\gamma \in [0, 1)$ is known as the *discount-rate parameter*. Define the *state-value function* $v_\pi : \mathcal{S} \to \mathbb{R}$ for policy π, the target for all the approximations we study in this article, as the expected return from each state given that actions are taken according to π:

$$v_\pi(s) \doteq \mathbb{E}[G_t \mid S_t = s, A_{t:\infty} \sim \pi] , \qquad \forall s \in \mathcal{S}. \tag{2}$$

To solve the MDP is to find an optimal policy π^*, defined as a policy that maximizes the value function from each state:

$$\pi^* \doteq \operatorname*{argmax}_{\pi} v_\pi(s), \qquad \forall s \in \mathcal{S}, \tag{3}$$

but in this article we do not address this problem directly, but instead focus on *policy evaluation*, the computation or estimation of v_π for a given policy π. Policy evaluation is a key subproblem underlying almost all efficient solution strategies for MDPs. In particular, we seek an approximation to the state-value function,

$$v_\theta(s) \approx v_\pi(s), \qquad \forall s \in \mathcal{S}, \tag{4}$$

where $\theta \in \mathbb{R}^n$, with $n \ll |\mathcal{S}|$, is the weight/parameter vector. The approximate value function can have arbitrary form as long as it is everywhere differentiable with respect to the weights. For example, it could be a cubic spline, or it could implemented by a multi-layer neural network where θ is the concatenation of all the connection weights. Henceforth we refer to θ exclusively as the weights, or weight vector, and reserve the word "parameter" for things like the discount-rate parameter, γ, and step-size parameters.

An important special case is that in which the approximate value function is linear in the weights and in features of the state:

$$v_\theta(s) \doteq \theta^\top \phi(s), \tag{5}$$

where the $\phi(s) \in \mathbb{R}^n$, $\forall s \in \mathcal{S}$, are feature vectors characterizing each state s, and $x^\top y$ denotes the inner product of two vectors x and y. The linear case is much better understood and we will focus on it for the next four sections.

3. Objectives for linear value-function approximation

Although parametric function approximation has been used almost from the beginnings of both DP and TDL (e.g., see Bellman & Dreyfus 1959, Samuel 1959, Sutton 1988), it has proven difficult to obtain fully satisfactory algorithms and theory. Even for the linear case the issues are not clear, and even the question of what kind of approximation we seek remains without a single concensus answer. Given a linear form for v_θ (5), what is the objective for θ? What is its best value? There are at least three important possible answers to this question, as we discuss in the three subsections below.

First, let us define the general term "value function" as any function from the MDP's state space $\mathcal{S}$ to the real numbers. For now let us assume that the state space is discrete, $\mathcal{S} = \{1, 2, \ldots, |\mathcal{S}|\}$, in which case a value function can be thought of as a real-valued vector of $|\mathcal{S}|$ components. We can distinguish the large space of all possible value functions from the smaller subspace of value functions that can be implemented by the linear function approximator at some value of θ, as suggested by Figure 2. We assume that the true value function v_π is too complex to be represented exactly as an approximation v_θ for any θ. Thus v_π is not in the subspace; in Figure 2, v_π is depicted as being above a planar subspace of representable functions.

A goal for approximation will generally include a weighting or distribution $d : \mathcal{S} \to \mathbb{R}$ specifying the degree to which we would like different states to be accurately valued. This

is new to approximation; no such distribution arises in the conventional theory for exactly solving discounted MDPs. If all state values can be exactly correct, there is no need to weight how they will be traded off, but with approximation there is. This distribution provides a natural norm and measure of distance in value-function space. For any value function v, define

$$\|v\| \doteq \sum_{s\in\mathcal{S}} d(s)v(s)^2. \tag{6}$$

(If the state space were continuous, then the sum would be replaced by an integral.) We will use only this d-weighted norm in this paper, so we do not explicitly indicate its dependence on d in our notation. The distance between two value functions v_1 and v_2 is then simply $\|v_1 - v_2\|$. For any value function v, the operation of finding the closest value function v_θ in the subspace of representable value functions is a projection operation. Formally, we define the projection operator $\Pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ as

$$\Pi v \doteq v_\theta \quad \text{where} \quad \theta = \arg\min_\theta \|v - v_\theta\| . \tag{7}$$

For a linear function approximator, the projection operator is linear, which implies that it can be represented as an $|\mathcal{S}| \times |\mathcal{S}|$ matrix:

$$\Pi \doteq \Phi \left(\Phi^\top D \Phi\right)^{-1} \Phi^\top D, \tag{8}$$

where D denotes the $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix with d on the diagonal, and Φ denotes the $|\mathcal{S}| \times n$ matrix whose rows are the feature vectors $\phi(s)^\top$, one for each state s:

$$D \doteq \begin{bmatrix} d(1) & & & 0 \\ & d(2) & & \\ & & \ddots & \\ 0 & & & d(|\mathcal{S}|) \end{bmatrix}, \qquad \Phi \doteq \begin{bmatrix} -\phi(1)^\top - \\ -\phi(2)^\top - \\ \vdots \\ -\phi(|\mathcal{S}|)^\top - \end{bmatrix}. \tag{9}$$

(Formally, the inverse in (8) may not exist, in which case the pseudoinverse is substituted.) Using these matrices, the squared norm of a vector can be written

$$\|v\| = v^\top D v, \tag{10}$$

and the approximate linear value function can be written

$$v_\theta = \Phi\theta. \tag{11}$$

3.1 Value error objective

The most obvious goal for approximation is simply to minimize the distance between the true and approximate value functions, which we call the *value error* (VE) objective:

$$J_{\text{VE}}(\theta) \doteq \|v_\pi - v_\theta\| . \tag{12}$$

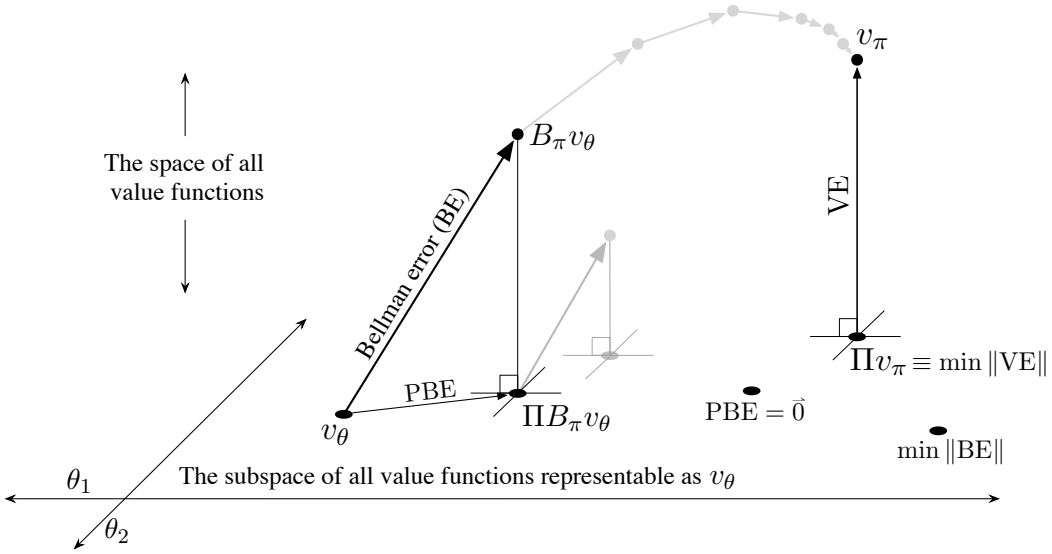


Figure 2: The geometry of linear value-function approximation. Shown as a plane here is the subspace of all functions representable by the function approximator. The three-dimensional space above it is the much larger space of all value functions (functions from $\mathcal{S}$ to $\mathbb{R}$). The true value function v_π is in this larger space and projects down to its best approximation in the value error (VE) sense. The best approximators in the BE and PBE senses are different and are also shown in the lower right. The Bellman operator takes a value function in the plane to one outside, which can then be projected back. If you could iteratively apply the Bellman operator outside the space (shown in gray above) you would reach the true value function, as in conventional DP.

The value function that minimizes this distance is, of course, Πv_π, the projection of the true value function into the subspace of representable functions, as shown in Figure 2.

To our knowledge, there is no practical deterministic algorithm for achieving this goal. The best methods are based on averaging over sample trajectories (a.k.a. rollouts) started according to d and evolving according to π and the MDP (which gives unbiased samples of $v_\pi(s)$). TDL algorithms achieve this goal in essentially the same way when they use maximally-long eligibility traces ($\lambda = 1$). All such Monte Carlo methods can be efficient if value estimates are needed for only a small part of the state space (i.e., if d is very concentrated) but tend to be inefficient (high variance) if the value function needs to be accurately approximated over a large portion of the state space. Beyond these practical considerations, it remains unclear whether achieving this goal would be better or worse than achieving one of the other two goals. We will not consider this goal further in this paper.

3.2 Bellman error objective

The second goal for approximation is to approximately solve the *Bellman equation* for policy π:

$$v_\pi = B_\pi v_\pi, \tag{13}$$

where $B_\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ is the *Bellman operator* for policy π, defined by

$$(B_\pi v)(s) \doteq \sum_{a \in \mathcal{A}} \pi(s,a) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v(s') \right], \qquad \forall s \in \mathcal{S},\ \forall v : \mathcal{S} \to \mathbb{R}, \tag{14}$$

which can also be written,

$$B_\pi v = r_\pi + \gamma P_\pi v, \qquad \forall v : \mathcal{S} \to \mathbb{R}, \tag{15}$$

where $r_\pi \in \mathbb{R}^{|\mathcal{S}|}$ is a vector whose entries give the expected immediate reward from each state under π, $[r_\pi]_s = \sum_{a \in \mathcal{A}} \pi(s,a) r(s,a)$, and $P_\pi \in \mathbb{R}^{|\mathcal{S}|} \times \mathbb{R}^{|\mathcal{S}|}$ is a state-transition matrix for policy π, with entries $[P_\pi]_{ji} = \sum_{a \in \mathcal{A}} \pi(i,a) p(j|i,a)$. The true value function v_π is the unique solution to the Bellman equation, and in this sense the Bellman equation can be viewed as an alternate way of defining v_π. For any value function v_θ not equal to v_π, we can ask the Bellman equation to hold approximately, $v_\theta \approx B_\pi v_\theta$. The error between the two sides of this equation we define as the *Bellman error* (BE):

$$\bar{\delta}_\theta \doteq B_\pi v_\theta - v_\theta. \tag{16}$$

The *Bellman error objective* is to minimize the norm of this vector:

$$J_{\mathrm{BE}}(\theta) \doteq \left\| \bar{\delta}_\theta \right\|, \tag{17}$$

Note that we cannot expect to drive $\bar{\delta}_\theta$ to zero if v_π is outside the representable subspace. Figure 2 shows the geometric relationships; note that the Bellman operator is shown as taking value functions inside the subspace outside to something that is not representable, and that the θ that minimizes BE is in general different from that which minimizes VE.

The BE was first proposed as an objective function for DP by Schweitzer and Seidmann (1985). Baird (1995, 1999) extended it to TDL based on stochastic gradient descent, and Engel, Mannor, and Meir (2003) extended it to least squares ($O(n^2)$) methods known as Gaussian Process TDL. In the literature, BE minimization is often referred to as Bellman residual minimization.

3.3 Projected Bellman error

The third goal for approximation is to approximately solve the *projected* Bellman equation:

$$v_\theta = \Pi(B_\pi v_\theta). \tag{18}$$

Unlike the original Bellman equation, the projected Bellman equation can be solved exactly for linear function approximators. The original TDL methods (Sutton 1988, Dayan 1992) converge to this solution, as does least-squares TDL (Bradke & Barto 1996, Boyan 1999). The goal of achieving (18) exactly is common; less common is to consider approximating it as an objective. Early work on gradient-TD (e.g., Sutton et al. 2009) appears to have been the first to explicitly propose minimizing the d-weighted norm of the error in (18), which we here call the *projected Bellman error* (PBE) objective:

$$J_{\mathrm{PBE}}(\theta) \doteq \left\| \Pi \bar{\delta}_\theta \right\|. \tag{19}$$

An intuitive understanding of this objective can be obtained by looking at the left side of Figure 2. Starting at v_θ, the Bellman operator takes us outside the subspace, and the projection operator takes us back into it. The between between where we end up and where we started is the PBE. An exact solution is when this vector is $\vec{0}$ and $J_{\text{PBE}} = 0$.

A more thorough understanding can be obtained by expanding and rewriting the PBE objective in matrix terms:

$$\begin{aligned} J_{\text{PBE}}(\theta) &= \left\|\Pi\bar{\delta}_\theta\right\|^2 \\ &= \bar{\delta}_\theta^\top \Pi^\top D \Pi \bar{\delta}_\theta \\ &= \bar{\delta}_\theta^\top D\Phi\left(\Phi^\top D\Phi\right)^{-1}\Phi^\top D\bar{\delta}_\theta \end{aligned} \tag{20}$$

(using the identity $\Pi^\top D\Pi = D\Phi\left(\Phi^\top D\Phi\right)^{-1}\Phi^\top D$)

$$= \left(\Phi^\top D\bar{\delta}_\theta\right)^\top\left(\Phi^\top D\Phi\right)^{-1}\left(\Phi^\top D\bar{\delta}_\theta\right). \tag{21}$$

The middle factor is the pseudoinverse of a symmetric, positive definite matrix which we will often denote simply as

$$C \doteq \Phi^\top D\Phi. \tag{22}$$

The inverse of this matrix plays a key role as a metric in value function space; notice that it also appears in the matrix representation of the norm (8). The first and third factors of the expression above for J_{PBE} are the same vector transposed; J_{PBE} will be zero exactly when this key vector is zero:

$$\begin{aligned} \vec{0} &= \Phi^\top D\bar{\delta}_\theta \\ &= \Phi^\top D\left(r_\pi + \gamma P_\pi\Phi\theta - \Phi\theta\right) \\ &= \Phi^\top Dr_\pi - \Phi^\top D(I - \gamma P_\pi)\Phi\theta \\ &= \quad b \quad - \quad A\theta, \end{aligned} \tag{23}$$

where

$$b \doteq \Phi^\top Dr_\pi \qquad \text{and} \qquad A \doteq \Phi^\top D(I - \gamma P_\pi)\Phi. \tag{24}$$

The matrix A is non-singular (Sutton 1988, Tsitsiklis & Van Roy 1997), and so this equation always has a solution,

$$\theta^*_{\text{PBE}} \doteq A^{-1}b \tag{25}$$

at which $J_{\text{PBE}} = 0$.

Finally, note that in the new notation, J_{PBE} can be written

$$J_{\text{PBE}}(\theta) = (A\theta - b)^\top C^{-1}(A\theta - b). \tag{26}$$

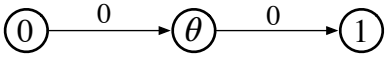
3.4 The Bellman error controversy

There is a small controversy in the field as to whether the BE objective or PBE objective is the most appropriate for value function approximation. It was originally argued that the BE objective was not appropriate (see Dayan 1992, Werbos 1990) and we have recently lent support to that argument (Sutton et al. 2009). However, Baird (1995, 1999) strongly supported BE minimization, and in recent years it has become popular with many researchers,

most notably in gaussian process TDL and kernel-based TDL (..., see particularly Scherrer 2010) We cannot properly resolve this controvery here—that would probably take a new theoretical result or extensive empirical comparisons, which would take us far afield. Nevertheless, in this paper we focus on the PBE objective. It is appropriate, then, that we motivate a little further our preference for the PBE objective over the seeming similar BE objective. Ultimately, one may remain agnostic as to which of these is best and simply take our results as the development of the PBE approach.

One of the best arguments against using the BE objective, and one that has not previously been made in the literature, is that it can not be estimated from data. This limits the BE objective to model-based settings, as in classical DP, or to simulators in which the state can be reset; it cannot be used for learning from a single sample trajectory. We discuss this in a later section.

The term "forward bootstrapping" is sometimes used to describe the way the state values in TDL are updated toward the values of the states that follow them. *Backward* bootstrapping is then updating toward the value of preceding states. The PBE objective encourages forward bootstrapping only whereas the BE objective encourages bootstrapping in both directions. A pure example of this difference between the objectives is given by the following small MDP:



Here the middle state is given by the scalar weight θ while the first and last states have fixed values of 0 and 1. (It is perfectly permissable for the approximator to assign fixed values to some states.) The only question then, is what value to give to the center state? Note that it is a free choice and can be done, by construction, without affecting the value of any other state. Minimizing the BE and PBE objectives gives it different values. To minimize the PBE objective, θ is set to 1, so that the state's value matches the state that follows it, whereas, to minimize the BE objective, θ is set to $\frac{1}{2}$ as a compromise between forward and backward bootstrapping. The BE objective favors a θ value halfway between the 0 and the 1 before and after it, splitting the Bellman error into two pieces whose sum of squares is minimal. As a result, minimizing J_{BE} results in a form of smoothing.

4. Dynamic programming with linear value-function approximation

In this paper we are interested in bootstrapping methods for parametric policy evaluation that apply in both planning (DP) and learning (TDL) settings. We cover the planning case first, in this and the next section, because the algorithms are deterministic and thus easier to analyze. The TDL methods can be viewed as stochastic methods for approximating the DP planning algorithms. Moreover, and contrary to common understanding, the key challenges of off-policy training and nonlinear function approximation appear just as strongly in planning as they do in learning. In this section our methods for addressing these challenges are presented in the deterministic DP setting before we go on to TDL in later sections. As in the previous section, we restrict attention to the case of linear function approximation, postponing the nonlinear case until later (Section 7).

Consider the standard DP algorithm for policy evaluation, which involves iteratively updating a value function. The value function $v : \mathcal{S} \to \mathbb{R}$ is initialized arbitrarily and then

updated according to

$$v \leftarrow r_\pi + \gamma P_\pi v, \tag{27}$$

The resultant sequence of value functions converges to a unique fixpoint, which is the state-value function $v_\pi = r_\pi + \gamma P_\pi v_\pi$. This algorithm is not feasible for large state spaces, of course, as it is a vector update on a vector with as many components as there are states. In addition, there is a matrix-vector product, which nominally involves a sum over the state space, but in practice this is not a concern as the possible next states are typically highly concentrated.

4.1 Temporal-difference DP

To adapt DP for use with a parameterized function approximator, we must provide an algorithm for updating the weights that is in some way analogous to the DP algorithm (27). The following algorithm may not have ever been explicitly written down before, but in a sense underlies much past work in TDL. We consider it to be the natural, first way to generalize policy-evaluation DP to parametric function approximation. We call it temporal-difference DP (TDDP) because of its close relationship to linear TD(0), a common TDL algorithm:

$$\theta \leftarrow \theta + \alpha \Phi^\top D \bar{\delta}_\theta \tag{28}$$

$$= \theta + \alpha \sum_{s \in \mathcal{S}} d(s) \bar{\delta}_\theta(s) \phi(s), \tag{29}$$

where $\alpha > 0$ is a constant step-size parameter, and d is the weighting of states introduced in the previous section. Notice that the update to θ is proportional to $\Phi^\top D \bar{\delta}_\theta$, the key vector that we earlier established is zero when the J_{PBE} is zero (see Eqn. 20). Thus, the PBE solution is a fixpoint (those not necessarily a stable fixpoint) of this method.

At first glance, TDDP may seem no more feasible for large state spaces than tabular DP. If one literally does a sum over all states, then the update is not feasible for large state spaces. One way for TDDP to be practical is if d is concentrated on a feasibly small subset of the states, perhaps the center points of a grid over a continuous state space, or a finite selection of test states. More generally, a close approximation to TDDP can be obtained by sampling from d and performing the update with just a few states (and then doing more iterations). Sampling introduces variance and would require that the step-size parameter be reduced over time to obtain convergence. The computational expense of computing $\bar{\delta}_\theta(s)$ can also be reduced by sampling over the possible next states (again at the cost of variance). As more sampling is introduced, and the samples are taken from a single sample path, this algorithm comes closer to TDL, in particular, the TD(0) algorithm (Sutton 1988). But for now it is useful to stick with the explicit, perhaps impractical, TDDP algorithm to assess its strengths and weaknesses. For example, any weaknesses we uncover can be expected to be inherited by any stochastic approximation to it.

To understand the convergence of TDDP's iteration, it is helpful to write it in terms of the key matrix A and the b vector (24) introduced in conjunction with the PBE objective. Using these it can be written simply as

$$\theta \leftarrow \theta - \alpha \left(A\theta - b \right). \tag{30}$$

From this it is clear that the value of b will participate in determining the fixpoint of the algorithm, but not at all in determining whether it will converge. Convergence of the iteration is completely determined by whether or not the matrix A is positive definite. If A is positive definite, then, for sufficiently small α, the iteration (30) is guaranteed to converge.

The matrix A has been proven to be positive definite if and only if the distribution d is an *on-policy distribution* (Tsitsiklis & Van Roy 1997, Sutton 1988), meaning any distribution that could be obtained in trajectories by following policy π. The trajectories can start anywhere, but once started must continue until an ending of some sort occurs. In general, in non-ergodic or episodic MDPs, there may be many on-policy trajectory distributions depending on the distribution of starting states. In our setting, without endings and assuming ergodicity, the only on-policy trajectory distribution is the stationary distribution under the policy, denoted

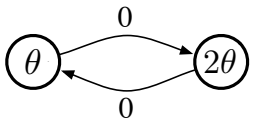
$$d_\pi(s) = \lim_{t\to\infty} \mathbf{Pr}\{S_t = s \mid S_0 = s_0, A_{0:t-1} \sim \pi\}, \tag{31}$$

which by assumption of ergodicity exists and is independent of s_0. Thus, linear TDDP is guaranteed to converge if and only if $d = d_\pi$.

When linear TDDP converges, it converges, of course, to a fixpoint of its update (28). We showed earlier (20) that this is exactly when J_{PBE} is zero. This remains true even in the off-policy case (when $d \neq d_\pi$).

That TDDP converges to a zero of the PBE for linear FA and the on-policy distribution is an important positive result, representing the most successful generalization ever of DP to function approximation. It was a breakthrough of sorts, representing significant progress towards addressing Bellman's "curse of dimensionality." However, linearity and the on-policy distribution remained significant limitations. In this paper we will present methods that remove both limitations, so let us examine them more carefully.

TDDP's limitation to the on-policy distribution appears more fundamental. Simple counterexamples were presented by Baird (1995), by Tsitsiklis and Van Roy (1997), and Scherrer (2011). Perhaps the simplest counterexample, and a good one for understanding the issues, is given by this MDP:



The states are labeled to characterize the function approximator, which assigns the first state a value of θ and the second a value of 2θ (theta is a scalar in this example). Note that this is a linear function approximator. Now suppose d puts all weight on the first state. Then the TDDP update reduces to

$$\theta \leftarrow \theta + \alpha(\gamma 2\theta - \theta) = (1 + \alpha(2\gamma - 1))\theta, \tag{32}$$

which diverges if $\theta \neq 0$ and $\gamma > \frac{1}{2}$. It is not important to this example that the second state is given zero weight; if it is given, say, 10% of the weight, then divergence still occurs (albeit at higher values of γ). Of course, if the second state is given equal weight, then that would be the on-policy distribution and convergence would be guaranteed.

So we see there is a fundamental sense in which DP does not work well with parametric function approximation. It is due to bootstrapping, the recursive update, not to TDL.

4.2 Gradient descent

TDDP converges reliably only when d is the on-policy distribution, and has no converge guarantees for nonlinear function approximators. In these senses, it is not a very robust algorithm, and neither are the corresponding TDL algorithms. This contrasts with function approximation in supervised learning settings, where methods based on gradient descent exhibit robust convergence to a local optimum for nonlinear function approximators and for all distributions. Gradient descent is a very general for designing robustly convergent learning algorithms, and it is natural to consider adapting it value function approximation.

A gradient-descent algorithm is one whose update is proportional to the negative gradient of some scalar function $J(\theta)$ over the weight-vector space, called the *objective function*. The objective typically represents some sort of error to be minimized, perhaps a squared error. Also typically, J has a minimal value, perhaps zero. The general schema for a gradient-descent algorithm is:

$$\theta \leftarrow \theta - \alpha \nabla J(\theta_t), \tag{33}$$

where α is a positive step-size parameter as before. As long as the objective satisfies basic smoothness conditions, algorithms of this form can be made convergent to a minimum of J simply by choosing the step-size parameter to be small. For sufficiently small α, the update must take θ downhill, to a smaller value of J. As J only decreases, yet is bounded below, convergence becomes inevitable. Gradient descent can also be done when the true gradient is not available, using instead an unbiased random estimate of the gradient; this is called stochastic gradient descent, and it also converges robustly (if the step-size parameter is reduced appropriately over time.

TDDP is not a gradient-descent algorithm for any function approximator (Barnard 1993).

4.3 Residual-gradient DP

The gradient-descent approach to parametric policy evaluation in TDL has been extensively explored using the BE objective, as previously noted. The natural DP algorithm for gradient descent in the BE objective is (Baird 1999, Sutton & Barto 1998):

$$\begin{aligned}
\theta &\leftarrow \theta - \frac{1}{2}\alpha \nabla J_{\mathrm{BE}}(\theta) \\
&= \theta - \frac{1}{2}\alpha \nabla \left[\bar{\delta}_\theta^\top D \bar{\delta}_\theta\right] \\
&= \theta - \alpha \left[\nabla \bar{\delta}_\theta\right]^\top D \bar{\delta}_\theta \\
&= \theta - \alpha \nabla [r_\pi + \gamma P_\pi \Phi\theta - \Phi\theta]^\top D \bar{\delta}_\theta \\
&= \theta - \alpha \left[\Phi - \gamma P_\pi \Phi\right]^\top D \bar{\delta}_\theta \\
&= \theta + \alpha \sum_{s\in\mathcal{S}} d(s) \bar{\delta}_\theta(s) \left(\nabla v_\theta(s) - \gamma \bar{\phi}^\pi(s)\right),
\end{aligned} \tag{34}$$

where the vector $\bar{\phi}^\pi(s) \in \mathbb{R}^{|\mathcal{S}|}$ is the expected next feature after s, under policy π,

$$\bar{\phi}^\pi(s) \doteq \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} p(s'|s,a)\phi(s'). \tag{35}$$

This algorithm, which we call residual-gradient DP (RGDP) because of its close relationship to Baird's (1995, 1999) residual-gradient TDL algorithm, can be approximated with $O(n)$ computation and memory and has good convergence properties. In particular, it converges to a local optimum of J_{BE} for nonlinear function approximators and for an arbitrary weighting distribution d. It does require a model or simulator, as has long been noted (but now we know this is true for any method that minimizes J_{BE}).

4.4 Gradient DP

We now present an algorithm based on gradient descent, like RGDP, but for J_{PBE}. We call the algorithm simply gradient DP, or GDP. For linear function approximation, it is defined by the following two updates, performed in this order:

$$\theta \leftarrow \theta + \alpha\Phi^\top D\bar{\delta}_\theta - \alpha\gamma(P_\pi\Phi)^\top D\Phi w_\theta \tag{36}$$

$$= \theta + \alpha \sum_{s\in\mathcal{S}} d(s)\left(\bar{\delta}_\theta(s)\phi(s) - \gamma\big(w^\top\phi(s)\big)\bar{\phi}^\pi(s)\right) \tag{37}$$

and

$$w \leftarrow w + \beta\Phi^\top D(\bar{\delta}_\theta - \Phi w) \tag{38}$$

$$= w + \beta \sum_{s\in\mathcal{S}} d(s)\left(\bar{\delta}_\theta(s) - w^\top\phi(s)\right)\phi(s), \tag{39}$$

where $w \in \mathbb{R}^n$ is a second weight vector, and $\beta > 0$ is a second step-size parameter. To obtain convergence, we assume that the w iteration is faster than the θ iteration ($\alpha \ll \beta$). For a fixed θ and sufficiently small β, w converges deterministically to

$$w_\theta = \big(\Phi^\top D\Phi\big)^{-1}\Phi^\top D\bar{\delta}_\theta. \tag{40}$$

We now show that, at this value for w, the main linear GDP update for θ (36) is gradient descent in the PBE:

$$\begin{aligned}
\theta &\leftarrow \theta - \alpha\frac{1}{2}\nabla J_{\text{PBE}}(\theta)\\
&= \theta - \alpha\frac{1}{2}\nabla\left[\bar{\delta}_\theta^\top D\Phi\big(\Phi^\top D\Phi\big)^{-1}\Phi^\top D\bar{\delta}_\theta\right]\\
&= \theta - \alpha[\nabla\bar{\delta}_\theta]^\top D\Phi\big(\Phi^\top D\Phi\big)^{-1}\Phi^\top D\bar{\delta}_\theta\\
&= \theta - \alpha\nabla[r_\pi + \gamma P_\pi\Phi\theta - \Phi\theta]^\top D\Phi w_\theta\\
&= \theta + \alpha\left[\Phi - \gamma P_\pi\Phi\right]^\top D\Phi w_\theta.
\end{aligned}$$

(a gradient-based DP algorithm analogous to Sutton et al.'s (2009) GTD2)

$$\begin{aligned}
&= \theta + \alpha\Phi^\top D\Phi w_\theta - \alpha\gamma(P_\pi\Phi)^\top D\Phi w_\theta\\
&= \theta + \alpha\Phi^\top D\Phi\big(\Phi^\top D\Phi\big)^{-1}\Phi^\top D\bar{\delta}_\theta - \alpha\gamma(P_\pi\Phi)^\top D\Phi w_\theta\\
&= \theta + \alpha\Phi^\top D\bar{\delta}_\theta - \alpha\gamma(P_\pi\Phi)^\top D\Phi w_\theta,
\end{aligned}$$

which is the desired linear GDP update for θ (36) with w_θ in place of w. In this sense, GDP is a gradient descent algorithm and, accordingly, it is relatively straightforward to prove

that it converges for arbitrary d (next subsection) and for nonlinear function approximators (Section 7).

The update for w (39) is a standard gradient-descent rule, often called LMS (Least Mean Square), for estimating $\bar{\delta}_\theta(s)$ as $w^\top \phi(s)$. For stationary θ, it is well known that this update converges to w_θ (e.g., see Widrow and Stearns 1985). This update can also be derived as gradient descent in its mean-squared error:

$$\begin{aligned} w &\leftarrow w - \beta \frac{1}{2} \nabla_w \left\| \bar{\delta}_\theta - \Phi w \right\| \\ &= w - \beta \frac{1}{2} \nabla_w \Big[(\bar{\delta}_\theta - \Phi w)^\top D (\bar{\delta}_\theta - \Phi w) \Big] \\ &= w - \beta \nabla_w \big[\bar{\delta}_\theta - \Phi w \big]^\top D (\bar{\delta}_\theta - \Phi w) \\ &= w + \beta \Phi^\top D (\bar{\delta}_\theta - \Phi w), \end{aligned}$$

which is the linear GDP update for w (38), completing the derivation of linear GDP.

4.5 Convergence of linear GDP

In this subsection we prove the convergence of linear GDP. GDP is a deterministic algorithm, and we have shown its close relationship to gradient descent in the previous subsection. Unfortunately, we cannot use standard results for gradient descent to prove convergence of GDP because the two updates (for θ and w) operate simultaneously, and neither is an exact gradient descent algorithm while the other is operating.

Theorem 1 (Convergence of linear GDP) *Consider the linear GDP algorithm given by* (36), (38), *and* (5), *for any distribution $d : \mathcal{S} \to \mathbb{R}^+$, for any finite MDP, for any bounded feature vectors $\phi : \mathcal{S} \to [-M, M]^n$, and with w initialized to zero. Then there exists a positive constant $\eta_{\min}$, such that, for any $\eta > \eta_{\min}$, there exists a positive constant $\alpha_{\max}$ such that, for any positive step-size parameters $\alpha < \alpha_{\max}$ and $\beta = \eta\alpha$, then J_{PBE} converges to zero. Moreover, $\eta_{\min}$ is the larger of 0 and the largest eigenvalue of the $n \times n$ symmetric matrix*

$$-C^{-1} \frac{A + A^\top}{2}, \tag{41}$$

where C and A are defined by (22) *and* (24).

Proof Linear GDP's two iterations (eqs. 36 and 38) can be rewritten as a single iteration in a combined parameter vector with $2n$ components, $z^\top = (w^\top, \theta^\top)$, and a new reward-related vector g, also with $2n$ components, as follows:

$$z \leftarrow z + \alpha \left(-Gz + g \right), \tag{42}$$

where

$$G \doteq \begin{pmatrix} \eta C & \eta A \\ C - A^\top & A \end{pmatrix}, \quad g \doteq \begin{pmatrix} \eta b \\ b \end{pmatrix},$$

where b is defined by (24). The convergence of z is entirely determined by the real parts of the eigenvalues of the matrix $I - \alpha G$. If they are all positive and less than 1, then z

converges to the fixed point satisfying $-Gz + g = 0$. Thus, we need only establish that: (i) the real parts of the eigenvalues of G are all positive; (ii) Choose α small enough that the largest real-part eigenvalue of αG is less than 1; (iii) Show that if $-Gz + g = 0$ then $J_{\text{PBE}} = 0$

The eigenvalues of G can be found by solving the characteristic equation, $\det(G-\lambda I) = 0$ (where $\det()$ denotes determinant). We have:

$$0 = \det(G - \lambda I) = \det \begin{pmatrix} \eta C_{\lambda,\eta} & \eta A \\ C - A^\top & A - \lambda I \end{pmatrix}, \quad \text{where } C_{\lambda,\eta} \doteq C - \frac{\lambda}{\eta} I.$$

This can be simplified using the determinant rule for partitioned matrices. According to this rule, if $A_1 \in \mathbb{R}^{n_1 \times n_1}, A_2 \in \mathbb{R}^{n_1 \times n_2}, A_3 \in \mathbb{R}^{n_2 \times n_1}, A_4 \in \mathbb{R}^{n_2 \times n_2}$ then for $X = [A_1 A_2; A_3 A_4] \in \mathbb{R}^{(n_1+n_2)\times(n_1+n_2)}, \det(X) = \det(A_1)\det(A_4 - A_3 A_1^{-1} A_2)$. Thus, we have

$$\begin{aligned} 0 = \det(G - \lambda I) &= \det(\eta C_{\lambda,\eta}) \det\Big(A - \lambda I - (C - A^\top)(\eta C_{\lambda,\eta})^{-1} \eta A\Big) \\ &= \eta^{2n} \det(C_{\lambda,\eta}) \det\Big(A - \lambda I - (C - A^\top) C_{\lambda,\eta}^{-1} A\Big). \end{aligned}$$

Note, to avoid singularity issues, through out this paper, we use Moore-Penrose pseudo-inverse for inverting matrices. To simplify the determinant term further, we can re-write the matrix inside the second determinant as follows:

$$\begin{aligned} A - \lambda I - (C - A^\top) C_{\lambda,\eta}^{-1} A &= A - \lambda I - C C_{\lambda,\eta}^{-1} A + A^\top C_{\lambda,\eta}^{-1} A \\ &= -\lambda I - \frac{\lambda}{\eta} C_{\lambda,\eta}^{-1} A + A^\top C_{\lambda,\eta}^{-1} A \\ &= (-\lambda A^{-1} C_{\lambda,\eta} + A^\top - \frac{\lambda}{\eta} I) C_{\lambda,\eta}^{-1} A \\ &= A^{-1}(-\lambda C_{\lambda,\eta} + A(A^\top - \frac{\lambda}{\eta} I)) C_{\lambda,\eta}^{-1} A, \end{aligned}$$

where we have used the identity $C C_{\lambda,\eta}^{-1} = I + \frac{\lambda}{\eta} C_{\lambda,\eta}^{-1}$ with $X(Y+X)^{-1} = (Y + X - Y)(Y + X)^{-1} = I - Y(Y+X)^{-1}$. Thus, using the determinant rule $\det(XY) = \det(X)\det(Y)$ for the two matrix X and Y, and also $\det(X^{-1}) = \det(X)^{-1}$, we have:

$$\begin{aligned} 0 = \det(G - \lambda I) &= \eta^{2n} \det(C_{\lambda,\eta}) \det\Big(A^{-1}(-\lambda C_{\lambda,\eta} + A(A^\top - \frac{\lambda}{\eta} I)) C_{\lambda,\eta}^{-1} A\Big) \\ &\quad \eta^{2n} \det(C_{\lambda,\eta}) \det(A^{-1}) \det\Big((-\lambda C_{\lambda,\eta} + A(A^\top - \frac{\lambda}{\eta} I))\Big) \det(C_{\lambda,\eta}^{-1} A) \\ &= \eta^{2n} \det(-\lambda C_{\lambda,\eta} + A(A^\top - \frac{\lambda}{\eta} I)), \end{aligned}$$

which implies that all the eigenvalues of matrix G satisfies (note, $\eta > 0$):

$$\det(-\lambda C_{\lambda,\eta} + A(A^\top - \frac{\lambda}{\eta} I)) = 0.$$

As such, there must exist a nonzero vector $x \in \mathbb{C}^n$, such that for any λ satisfying above, we have

$$x^* \left(-\lambda C_{\lambda,\eta} + A(A^\top - \frac{\lambda}{\eta} I)\right) x = 0,$$

where x^* is the complex conjugate of vector x (including its transpose), and $x^*x = \|x\|^2 > 0$. Using the definition of $C_{\lambda,\eta}$ in the above equation, would lead us to the following quadratic equation in terms of λ,

$$\|x\|^2\lambda^2 - (\eta x^*Cx + x^*Ax)\lambda + \eta\|Ax\|^2 = 0,$$

where $\|Ax\|^2 = x^*A^\top Ax = x^*AA^\top x$. This quadratic equation has two solutions, λ_1 and λ_2, where [1]

$$\lambda_1\lambda_2 = \frac{\eta\|Ax\|^2}{\|x\|^2} > 0, \quad \lambda_1 + \lambda_2 = \frac{(\eta x^*Cx + x^*Ax)}{\|x\|^2}.$$

Because the product of the two solutions, $\lambda_1\lambda_2$, is a positive and real number, therefore, $\lambda_2 = \kappa\lambda_1^*$, where $\kappa \in \mathbb{R}^+$. Thus, we have $\lambda_1 + \lambda_2 = \lambda_1 + \kappa\lambda_1^*$. Let $\mathrm{Re}(\lambda)$ denote the real-part of λ.

Hence, due to $\mathrm{Re}(\lambda_1+\lambda_2) = (1+\kappa)\mathrm{Re}(\lambda_1) = (1+1/\kappa)\mathrm{Re}(\lambda_2)$, if we show $\mathrm{Re}(\lambda_1+\lambda_2) > 0$ for some conditions on η, that would imply $\mathrm{Re}(\lambda_1) > 0$ and $\mathrm{Re}(\lambda_2) > 0$.

Let us write $\mathrm{Re}(\lambda_1+\lambda_2) > 0$ in the following form, using the indentity $\mathrm{Re}\,(\lambda) = (\lambda+\lambda^*)/2$ and the fact that A and C are matrices whose elements are real numbers:

$$\begin{aligned}
\mathrm{Re}(\lambda_1 + \lambda_2) = \frac{\mathrm{Re}\,(\eta x^*Cx + x^*Ax)}{\|x\|^2} &= \frac{(\eta x^*Cx + x^*Ax) + (\eta x^*Cx + x^*Ax)^*}{2\|x\|^2} \\
&= \frac{\left(2\eta x^*Cx + x^*(A + A^\top)x\right)}{2\|x\|^2} \\
&> 0,
\end{aligned}$$

which is equivalent to

$$2\eta x^*Cx + x^*(A + A^\top)x > 0.$$

Thus, $\eta > -x^*(A+A^\top)x/(2x^*Cx)$. Clearly, the above inequality holds, for all λ eigenvalues, if we choose η such that

$$\eta > \max_{z\neq 0,\ z\in\mathbb{C}^n} \frac{-z^\top Hz}{z^\top Cz}, \quad H \doteq \frac{A + A^\top}{2}.$$

Here, the H and C matrices are symmetric, hence their eigenvalues and eigenvectors are reals. For any $z \neq 0$, $z \in \mathbb{R}^n$, let $y = C^{1/2}z$. Then when $z^\top Cz = 1$, we also have $\|y\|^2 = 1$. Therefore, it suffices to have

$$\eta > \max_{\|y\|^2=1} y^\top \left(-C^{-1/2}HC^{-1/2}\right) y,$$

which is equivalent to $\eta > \lambda_{\max}\left(-C^{-1/2}HC^{-1/2}\right)$. Note, $C^{-1/2}HC^{-1/2}$ is symmetric, so its eigenvalues are all real. Also, because

$$\lambda_{\max}\left(-C^{-1/2}HC^{-1/2}\right) = \lambda_{\max}\left(-C^{-1}H\right),$$

1. The quadratic equation, $ax^2 + bx + c = 0$ (with $a \neq 0$), has two solutions. The product and the sum of these two solutions are, $x_1x_2 = \frac{c}{a}$ and $x_1 + x_2 = \frac{-b}{a}$, respectively.

then, because η is positive number, we conclude

$$\eta > \eta_{\min} = \max\{0, \lambda_{\max}\left(-C^{-1}H\right)\},$$

which suffices to guarantee that all the real-part eigenvalues G are positive. The next step is to choose α such that it is smaller than the inverse of largest real-part eigenvalue of G. The latter condition makes the z-iterate convergent to the fixed point which satisfies $-Gz + g = 0$.

Finally, we show that if $-Gz + g = 0$ then $J_{\text{PBE}} = 0$. Because $-A\theta + b = 0$ implies $J_{\text{PBE}} = 0$, then we are done if we show that the θ-fixed point of iterate z satisfies, $-A\theta + b = 0$. To show this, observe that $-Gz + g = 0$, in terms of w and θ, can be written in the following form:

$$(-A\theta + b) - (C - A^\top)w = 0, \text{ and } \quad -Cw + (-A\theta + b) = 0.$$

From the above linear equations we conclude $-A\theta + b = 0$, thus finishing the proof. ■

4.6 Empirical counterexamples for off-policy d

In this section we demonstrate empirically that TDDP diverges, and linear GDP converges, on the small counterexample given at the beginning of this section and on Baird's counterexample.

The small example given earlier is shown inset in Figure **??**, which also shows the divergence of θ under TDDP and the convergence of both θ and w to zero under GDP. In this example, θ was initialized to 1, w was initialized to zero, and d was 0.9 for the first state and 0.1 for the second. The step-size parameters were $\alpha =?$ and $\beta =?$. Notice the way w has to first become nonzero before it can bring θ, along with itself, back to zero.

Baird's (1995) counterexample is shown in Figure 3. The Markov decision process (MDP) is depicted in Fig. 3. The reward is zero in all transitions, thus the true value functions for any given policy is zero. The behavior policy, in this example, chooses the solid line action with probability of 1/7 and the dotted line action with probability of 6/7. The goal is to learn the value of a target policy that chooses the solid line more often than the probability of 1/7. In this example, the target policy choose the solid action with probability of 1.

Value functions are approximated linearly. Both TD(0) and dynamic programming (with incremental updates), however, will diverge on this example; that is, their learning parameters will go to $\pm\infty$ as is illustrated in Figure 3.

Also freaky features goes in this section, maybe before Baird's counterexample.

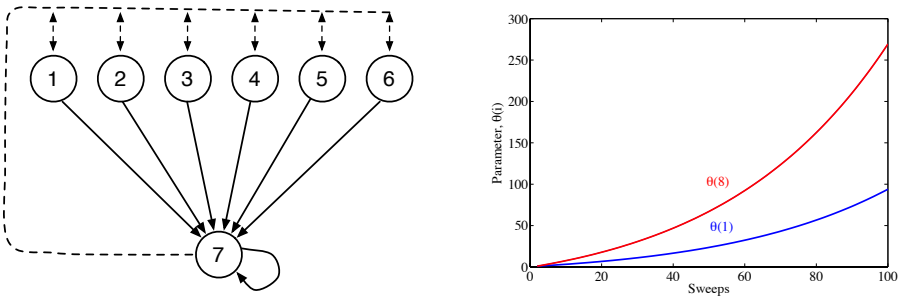


Figure 3: Left panel: Baird's counterexample (7 state version). Every transition in this MDP receives zero reward. Each state, has two actions, represented by solid line and dotted line. Solid line action only make transition to state state 7, while dotted line action uniformly make transition to one of states 1-6 with probability of 1/6.The value functions are approximated linearly in the form of $V(i) = 2\theta(i) + \theta_0$, for $i \in \{1, ..., 6\}$, and $V(7) = \theta(7) + 2\theta_0$. Here, the discount factor is $\gamma = 0.99$. The zero-PBE solution, in this example, is $\theta(i) = 0$, $i \in \{1, ..., 7\}$, and $\theta_0 = 0$. Right panel: The weights diverge to infinity under TDDP with all weights equally weighted.

5. Learning and identifiability

In learning, the MDP is viewed as a generator of a sample data path, an infinite-length trajectory of feature vectors, actions, and rewards: $\phi(S_0), A_0, R_1, \phi(S_1), A_1, R_2, \phi(S_2), \ldots$. The goal is to use this data to approximate the value function v_π for the *target policy,* π, while the actions are selected according to another given policy μ, known as the *behavior policy*. The two policies may be the same, the *on-policy* case, or they may be different, in which case the data and learning is said to be "off" the target policy, or *off-policy*. Note that the states $S_t, t = 0, 1, 2, \ldots$ are not in the trajectory, only the corresponding feature vectors $\phi(S_t)$. This is not a limitation, as the feature vectors could in principle capture the entire state, and allows us to include cases of partial observability. The policies, π and μ, are left unrestricted and may depend on the state arbitrarily. We do assume that at each step t, the action probabilities $\pi(S_t, a)$ and $\mu(S_t, a)$ are known for all $a \in \mathcal{A}$ without extensive computation. In this section we also assume linear function approximation (5), postponing the nonlinear case until Section 7.

The MDP and μ together are assumed to form an ergodic process with stationary distribution $d_\mu : \mathcal{S} \to \mathbb{R}^+$, defined as in (31) (if there are states that do not recur with positive probability under μ, then they can simply be removed from the state set). Let us assume that the initial state, S_0, is also sampled from d_μ. The probability distribution over random all variables S_t, A_t, and R_{t+1}, for $t = 0, 1, 2, \ldots$ is then completely defined, and we can talk unambiguously about the expectation of these random variables and other random variables defined in terms of these. Several examples of this are given in the second paragraph below.

Whereas earlier we considered planning (DP) objectives based on an arbitrary distribution d, in this section we consider learning objectives only for $d = d_\mu$. The difference is that in the planning case we have direct access to the states and can examine or sample them arbitrarily, whereas in the learning case the states are hidden; all we know about the states in the trajectory is that in the long run their proportion matches d_μ, even though we have no idea what d_μ is. In a later section we examine some important generalizations, but for now we assume $d = d_\mu$.

In off-policy learning we are often concerned with the ratio of taking an action under the target and behavior policies, sometimes called the *importance sampling ratio*:

$$\rho(s,a) = \frac{\pi(s,a)}{\mu(s,a)}. \tag{43}$$

The expectation of this ratio on state–action pairs encountered in the trajectory is

$$\mathbb{E}[\rho(S_t, A_t)] = \sum_s d_\mu(s) \sum_a \mu(s,a) \frac{\pi(s,a)}{\mu(s,a)} = \sum_s d_\mu(s) \sum_a \pi(s,a) = 1. \tag{44}$$

Note that we have not explicitly conditioned on μ (or the MDP) because of our convention for expectations that these are implicit. Next we follow this convention to express the key components of the PBE objective, C, A, and b, as expectations. First, to further simplify the notation, let us define $\rho_t \doteq \rho(S_t, A_t)$ and $\phi_t \doteq \phi(S_t)$. Then:

$$C = \mathbb{E}\left[\rho_t \phi_t \phi_t^\top\right], \; A = \mathbb{E}\left[\rho_t \phi_t \big(\phi_t - \gamma \phi_{t+1}\big)^\top \;\middle|\; A_t \sim \pi\right], \text{ and } b = \mathbb{E}[\rho_t R_{t+1} \phi_t \mid A_t \sim \pi]. \tag{45}$$

Note that these expectations condition only on actions, which are assumed visible in the trajectory, and not on states, which are not observed. As such, all of these expectations can be estimated from the data trajectory by averaging observable quantities. Recall that the PBE objective can be written in terms of C, A, and b, (eqn. 26) so the fact that these can be estimated from data means that J_{PBE} itself can be estimated from data. As we will show later, the gradient of J_{PBE} with respect to θ and the minimizing value of θ can also be determined from the data trajectory. This might seem like a small thing, but it turns out it is not true for either the VE or BE objectives. Neither of these can be estimated, or *identified*, from data, as we show next.

5.1 Identifiability

Let us consider more carefully the relationship between the MDP, the possible data trajectories, and the objectives of learning. As already described, the MDP and behavior policy together completely determine the probability distribution over data trajectories. Assume for the moment that the state, action, and reward sets are all finite. Then, for any finite sequence $\xi = \phi_0, a_0, r_1, \ldots, r_k, \phi_k$, there is a well defined probability (possibly zero) of it occuring as the initial portion of a trajectory, which we may denoted $\mathcal{P}_\mu(\xi) = \mathbf{Pr}\{\phi(S_0) = \phi_0, A_0 = a_0, R_1 = r_1, \ldots, R_k = r_k, \phi(S_k) = \phi_k\}$. The distribution $\mathcal{P}_\mu(\xi)$ then is a complete characterization of a source of data trajectories. To know $\mathcal{P}_\mu(\xi)$ is to know everything about the statistics of the data, but it is still less than knowing the MDP. In particular it is not enough to reconstruct or *identify* the MDP. Further, the

VE and BE objectives are readily computed from the MDP as described in Section 3, but they also are not identifiable and cannot be determined from $\mathcal{P}_\mu(\xi)$ alone.

The possible dependency relationships among the data distribution, MDPs, and various objectives are summarized in Figure 4. The left side of the figure treats the non-bootstrapping objective, J_{VE} (12). It indicates that two different MDPs, MDP_1 and MDP_2, can produce the same data distribution, yet have different VEs. The simplest example of this is the two MDPs shown below:

These are MDPs with only one action from each state, so they are in effect Markov chains. Where two edges leave a state, both possibilities are assumed to occur with equal probability. The numbers indicate the reward received on each edge traversal. All the states appear the same; they all produce the same feature vector $\phi = 1$ and have approximated value θ, a scalar. Thus, the only varying part of a data trajectory is the rewards. The left MDP stays in the same state and emits an endless stream of 0s and 2s i.i.d. at random, each with 50-50 probability. The right MDP, on every step, either stays in its current state or switches to the other, with 50-50 probability. The reward is deterministic in this MDP, always a 0 from one state and always a 2 from the other, but because the state is i.i.d. 50-50, the observable data is again an endless stream of 0s and 2s at random, identical to that produced by the left MDP. Thus, two different MDPs can produce the same data distribution as shown in the figure. This proves that the relationship between MDPs and data distributions is many-to-one and not invertible. We say that the MDP is *not identifiable*, meaning that it is not a function of the observable data distribution, and thus in principle cannot be determined from data.

This pair of MDPs demonstrate that J_{VE} is also not identifiable. Let $\gamma = 0$ and $\theta = 1$. Then the true values of the three states are 1, 0, and 2, left to right, and the J_{VE} of the left MDP is 0 while the J_{VE} of the right MDP is 1, for any d. Thus, the J_{VE} is different for two MDPs with the same data distribution and the J_{VE} cannot be determined from data. There is a saving grace, however. Even though the two J_{VE}s can be different, the value of θ that minimizes them is always the same and can always be determined by minimizing another objective, based on the return error (RE), which is identifiable, as shown in the figure. The RE objective is the mean-squared error between the approximate values and what the returns would be under the target policy:

$$J_{\text{RE}}(\theta)^2 = \mathbb{E}\Big[\big(v_\theta(S_t) - G_t\big)^2 \;\Big|\; A_{t:\infty} \sim \pi\Big]. \tag{46}$$

It is not difficult to show that

$$J_{\text{RE}}(\theta)^2 = J_{\text{VE}}(\theta)^2 + \mathbb{E}\Big[\big(v_\pi(S_t) - G_t\big)^2 \;\Big|\; A_{t:\infty} \sim \pi\Big], \tag{47}$$

where the second term does not depend on θ, but only on characteristics of π and the MDP. Thus, if one finds the minimal θ for J_{RE}, then one will also have found the minimum for J_{VE}, even though J_{VE} itself is not identifiable. The RE objective is identifiable as it is a

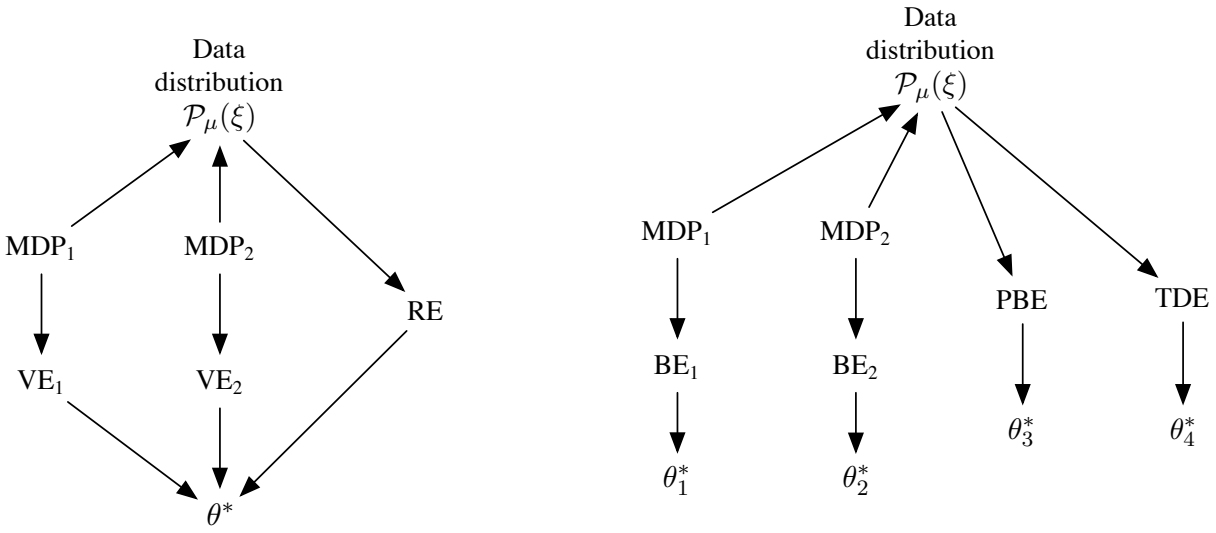


Figure 4: Causal relationships among the data distribution, MDPs, and errors for non-bootstrapping (left) and bootstrapping (right) objectives. In both cases, two different MDPs can produce the same data distribution. For non-bootstrapping objectives, the VE can be different for the two MDPs, and thus is not identifiable, but the optimal weights are the same and can be determined by optimizing the RE objective, which is identifiable. For bootstrapping objectives, both the BE and its optimum can be different for the two MDPs, and they have no coincidence with the identifiable errors, PBE and TDE, or their optima. Thus, minimizing J_{BE} is not a feasible objective for learning.

function only of the data distribution and the two policies. In the on-policy case at least, the RE objective can also be estimated easily from the data (the off-policy case is probably also possible using importance sampling techniques (e.g., Precup & Sutton 2000)).

But let us return to the bootstrapping objectives, J_{BE} and particularly J_{PBE}, which are of primary interest in this article. The dependencies here are summarized in the right half of Figure 4. To show the full range of possibilities we need a slightly more complex example than that considered above. Consider the following two MDPs:

The MDP on the left has two states that are represented distinctly; each has a separate weight so that they can take on any value. The MDP on the right has three states, two of which, B and B′, are represented identically and must be given the same approximate value. We can imagine that θ has two components and that the value of state A is given by the first component and the value of B and B′ is given by the second. Notice that the observable

data is identical for the two MDPs. In both cases the agent will see single occurrences of A (or rather the feature vector for A) followed by a 0, then some number of Bs, each followed by a -1 except the last which is followed by a 1, then we start all over again with a single A and a 0, etc. All the statistical details are the same as well; in both MDPs, the probability of a string of k Bs is 2^{-k}.

Now consider the value function $v_\theta = \vec{0}$. In the first MDP, this is an exact solution, and J_{BE} is zero. In the second MDP, this solution produces a squared error in both B and B′ of 1, such that $J_{\text{PBE}} = \sqrt{d(\mathsf{B})1 + d(\mathsf{B}')1} = \sqrt{2/3}$ if $d = d_\mu$. As with the previous example (there for the VE objective), these two MDPs, which generate the same data, have different J_{BE}s.

In this case, however, there is no saving grace; the minimizing value of θ is also different for the two MDPs For the first MDP, the minimal-J_{BE} value function is the exact solution $v_\theta = \vec{0}$ for any γ. For the second MDP, the minimum-J_{BE} value function is a complicated function of γ, but in the limit, as $\gamma \to 1$, it is $v_\theta(\mathsf{B}) = v_\theta(\mathsf{B}') = 0$ and $v_\theta(\mathsf{A}) = -\frac{1}{2}$. Thus the value function that minimizes J_{BE} cannot be estimated from data alone; knowledge of the MDP beyond what is revealed in the data is required. In this sense, it is impossible in principle to pursue the BE objective from data alone.

It may be surprising that the J_{BE}-minimizing value of A is so far from zero. Recall that A has a dedicated weight and thus its value is unconstrained by function approximation. A is followed by a reward of 0 and transition to a state with a value of nearly 0, which suggests $v_\theta(\mathsf{A})$ should be 0; why is its optimal value substantially negative rather than 0? The answer is that making the value of A negative reduces the error upon arriving in A from B. The reward on this deterministic transition is 1, which implies that B should have a value 1 more than A. Since $v_\theta(\mathsf{B})$ is approximately zero, $v_\theta(\mathsf{A})$ is driven toward -1. The J_{BE}-minimizing value of $v_\theta(\mathsf{A}) \approx -\frac{1}{2}$ is a compromise between reducing the errors on leaving and on entering A. Alternatively, one could seek to minimize only the error on leaving states; this is what happens with the PBE objective.

5.2 The TDE objective

As we have shown, J_{BE} and its minimum are not identifiable from data or even from complete knowledge of $\mathcal{P}_\mu$. They cannot be estimated without taking multiple samples from the same state, which requires a model/simulator. This may be a bit surprising, as it is natural to think of J_{BE} as an expectation. It is the expectation of the square of the Bellman error:

$$J_{\text{BE}}(\theta)^2 \doteq \left\|\bar{\delta}_\theta\right\| = \mathbb{E}\left[\bar{\delta}_\theta(s)^2\right], \tag{48}$$

and the Bellman error itself can be written as an expectation:

$$\bar{\delta}_\theta(s) = \mathbb{E}[\delta_t \mid S_t = s, A_t \sim \pi] = \mathbb{E}[\rho_t \delta_t \mid S_t = s] \tag{49}$$

where δ_t is a random variable, dependent on θ, known as the TD error (TDE).

$$\delta_t \doteq R_{t+1} + \gamma v_{\theta_t}(S_{t+1}) - v_{\theta_t}(S_t). \tag{50}$$

Unfortunately, if (48) and (49) are combined, the result is an expectation of an expectation, which cannot be sampled or otherwise estimated from data as we have shown. If one ignores

the second expectation, but simply uses δ_t directly, one obtains the TDE objective:

$$J_{\text{TDE}}(\theta)^2 \doteq \mathbb{E}\left[\delta_t^2 \;\middle|\; A_t \sim \pi\right] = \mathbb{E}\left[\rho_t \delta_t^2\right].$$

This objective is identifiable and can be minimized from data, but of course the minimum is not the same as that of J_{BE}.

5.3 Multi-step bootstrapping objectives

[This is where we introduce lambda-generalizations of the PBE. Or maybe do all the multi-step stuff together, later.]

We define the multi-step bootstrapping Bellman operator, B_π^λ, according to

$$(B_\pi^\lambda v)(s) \doteq \sum_{a\in\mathcal{A}} \pi(s,a)\left[r(s,a) + \gamma(s)\sum_{s'\in\mathcal{S}} p(s'|s,a)\left[(1-\lambda(s'))v(s') + \lambda(s')(B_\pi^\lambda v)(s')\right]\right], \tag{51}$$

where $\gamma(s) : \mathcal{S} \mapsto [0,1]$ is state-dependent discount factor, and $\lambda : \mathcal{S} \mapsto [0,1]$ is a state dependent bootstrapping parameter.

Now the multi-step bootstrapping J_{PBE} is

$$J_{\text{PBE}}(\theta,\lambda) \doteq \left\|v_\theta - \Pi B_\pi^\lambda v_\theta\right\| = \left\|\Pi \bar{\delta}_\theta^\lambda\right\|, \tag{52}$$

where $\bar{\delta}_\theta^\lambda$ is multi-step bootstrapping Bellman error (also called $\lambda-$ weighted Bellman error)

$$\bar{\delta}_\theta^\lambda \doteq B_\pi^\lambda v_\theta - v_\theta. \tag{53}$$

[since we are doing DP here maybe we don't need to introduce forward-backward view here?]

5.4 Action values

Maybe we should do all the action-value stuff later, together.

It is appropriate to introduce this here because it is primarily in a learning setting that our goal becomes to approximate action values rather than state values. Perhaps we will be able to formulate this in such a way that we can handle state values and action values simultaneously. But we should not press that if it is looking tricky or complicated.

6. Gradient temporal-difference learning (GTD)

We are now ready to present our main learning algorithms. The first of these, linear GTD(0), is a one-step gradient-TD algorithm for learning an an approximate linear state-value function. It is defined by the following two iterations:

$$\theta_{t+1} = \theta_t + \alpha_t \rho_t \left[\delta_t \phi_t - \gamma \phi_{t+1}\big(\phi_t^\top w_t\big)\right], \tag{54a}$$

$$w_{t+1} = w_t + \beta_t \rho_t \big(\delta_t - \phi_t^\top w_t\big)\phi_t, \tag{54b}$$

where $\theta_t \in \mathbb{R}^n$ and $w_t \in \mathbb{R}^n$ are two modifiable weight vectors, $\alpha_t > 0$ and $\beta_t > 0$ are two sequences of step-size parameters, and ρ_t, ϕ_t, and δ_t are as defined in Section 5 with $v_{\theta_t}(s) = \theta_t^\top \phi(s)$. These updates are chosen so that in expectation the weight vectors θ_t and w_t evolve as θ and w do in GDP. That is, for any θ, w, α, and β, suppose $\alpha_t = \alpha$ and $\beta_t = \beta$, then

$$\begin{aligned}
\mathbb{E}[\theta_{t+1} - \theta_t \mid \theta_t = \theta, w_t = w] &= \mathbb{E}\Big[\alpha\rho_t\big[\delta_t\phi_t - \gamma\phi_{t+1}\big(\phi_t^\top w\big)\big] \;\Big|\; \theta_t = \theta\Big] \\
&= \alpha\mathbb{E}\Big[\rho_t\delta_t\phi_t - \gamma\rho_t\phi_{t+1}\big(\phi_t^\top w\big) \;\Big|\; \theta_t = \theta\Big] \\
&= \alpha\sum_s d(s)\Big[\bar{\delta}_\theta(s)\phi(s) - \gamma\bar{\phi}^\pi(s)\big(\phi(s)^\top w\big)\Big]
\end{aligned}$$

(using (48) and (35)), which is the linear GDP update for θ (37), and

$$\begin{aligned}
\mathbb{E}[w_{t+1} - w_t \mid \theta_t = \theta, w_t = w] &= \mathbb{E}\Big[\beta\rho_t\big(\delta_t - \phi_t^\top w\big)\phi_t \;\Big|\; \theta_t = \theta\Big] \\
&= \beta\mathbb{E}\Big[\rho_t\delta_t\phi_t - \rho_t\big(\phi_t^\top w\big)\phi_t \;\Big|\; \theta_t = \theta\Big] \\
&= \beta\sum_s d(s)\Big[\bar{\delta}_\theta(s)\phi(s) - \big(\phi_t^\top w\big)\phi_t\Big]
\end{aligned}$$

(using (48) and (44)), which is the linear GDP update for w (39).

The first step is to introduce a general linear stochastic update in the following form:

$$Z_{t+1} = Z_t + \alpha_t\left[-G(X_t)Z_t + g(X_t)\right], \tag{55}$$

where $G(X_t)$ and $g(X_t)$ are... So $G(X_t)$ matrix will become

$(X_t)_{t\geq 0}$ is a Markov Process with an underlying unique invariant distribution, and $G(X_t)$ and $g(X_t)$ are stochastic matrix and vector, respectively, as a function of process X_t.

Therefore the following expectations are well-defined $G \doteq \lim_{t\to+\infty}\mathbb{E}[G_t(X_t)]$, $g = \lim_{t\to+\infty}\mathbb{E}[g(X_t)]$. For brevity, here we denote $\mathbb{E}_\infty[.]$ to refer the expectation of random variable with respect to invariant distribution. Thus, we have

$$G = \mathbb{E}_\infty[G(X_t)], \quad g = \mathbb{E}_\infty[g(X_t)]. \tag{56}$$

where H is a constant matrix and h is a constant vector.

Then, we re-write the iterate() in the following form:

$$Z_{t+1} = Z_t + \alpha_t\left[-GZ_t + g + M(X_t, Z_t)\right], \tag{57}$$

where

$$M(X_t, Z_t) = -\left(G(X_t - G)\right)Z_t + \left(g(X_t) - g\right), \tag{58}$$

denotes the noise term.

Theorem 2 (Convergence of Linear Stochastic Iterate (55)) *Let $(X_t)_{t\geq 0}$ be a uniformly bounded sequence generated according to a Markov Process with a unique invariant stationary distribution. Assume that matrix G and vector g in (57) are well-defined and all the real-part eigenvalues of matrix G in (55) are positive.Then the stochastic iterate (55) converges to z_∞ with probability one, where z_∞ satisfies $Gz_\infty + g = 0$, if the following conditions hold*

(C1) *The determinisitc update, $z \leftarrow z+\alpha(Gz+g)$, is convergent, for a small enough positive α.*

(C2) *The process $(X_t)_{t\geq 0}$ is uniformly bounded, and thus is stable as $t \to +\infty$.*

(C3) *There exist scalars K and q such that*

$$\sum_{t=0}^{\infty} \| \mathbb{E}[M(X_t, z) \mid X_0 = x] \| \leq K(1 + \|z\|)(1 + \|x\|_2^q),$$

where $M(X_t, z)$ matrix is defined in (58). This condition can be seen as Mixing condition, that is, it implies $G(X_t)Z + g(X_t)$ approaches to $GZ + g$ at a sufficiently rapid rate.

(C4) *Considering $\nu(x, z) = \sum_{t=0}^{\infty} \mathbb{E}[M(X_t, z) \mid X_0 = x]$, then there exists K and q, such that $\|\nu(x, z) - \nu(x, z')\| \leq K\|z - z'\| (1 + \|x\|^q)$. This condition controls the growth of noise term as z increases.*

(C5) *$\sum_{t=0}^{\infty} \alpha_t = +\infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < +\infty$.*

Proof This theorem is a special case of Theorem 17 (page 239) of Benveniste et al. (1990) for stochastic linear systems. ■

now we need a corollary for the above theorem customized for GTD(0).

For example, for GTD(0), we can write $Z_t = (w_t, \theta_t)^\top$ and thus get the following special form for $H(X_t)$:

$$-\rho_t \begin{pmatrix} \eta\phi_t\phi_t^\top & \eta\phi_t \left(\phi - \gamma\phi_{t+1}\right)^\top \\ \gamma\phi_t\phi_t^\top & \phi_t \left(\phi_t - \gamma\phi_{t+1}\right)^\top \end{pmatrix},$$

$$g(X_t)^\top \doteq R_t(\eta e_t, e_t)^\top .$$

$$X_t = (S_t, A_t, R_t, S_{t+1}, \phi_t, \phi_{t+1}, \rho_t)$$

Z_t is another process and is initialized with an arbitrary value Z_0 and has the iterate () . Assume G has positive real-part eigenvalues. Assume M_{t+1} is Martingale noise and has the following properties.... The with step-size condition... Z_t converges to the solution .... w.p.1.

Let $\mathbb{E}_0[.]$ stand for the invariant distribution, therefore we can write $\mathbb{E}_0[G(X_t)] \doteq \lim_{t\to+\infty} \mathbb{E}[G(X_t)]$, and $\mathbb{E}_0[g(X_t)] \doteq \lim_{t\to+\infty} \mathbb{E}[g(X_t)]$, thus G and g are well defined. construct matrix G_{t+1} and vector g_{t+1} as follows:

$$G(X_t) \doteq \begin{pmatrix} -\eta\rho_t\phi_t\phi_t^\top & \eta\rho_t\phi_t(\gamma\phi_{t+1} - \phi_t)^\top \\ -\gamma\rho_t\phi_{t+1}\phi_t^\top & \rho_t\phi_t(\gamma\phi_{t+1} - \phi_t)^\top \end{pmatrix}, \quad g(X_t)^\top \doteq (\eta\rho_t R_t\phi_t^\top, \rho_t R_t\phi_t^\top) .$$

Let us consider random variable $X_t \doteq (S_t, A_t, R_{t+1}, \phi_t)$ which is generated according to a Markov process [why?]. where X_t is stationary sequence obtained according to a Markov Process? X_t has Markovian structure, that is, X_t only depends on X_{t-1}.

we need to say something about the way we prove convergence before the following lemma. Let's concatenate the updates and construct stochastic iterates based on matrix G. Then we want to say we want to say Markov- noise diminishes over time,

Lemma 1 (Martingale Noise Lemma for GTD(0)) *Let us consider random variable*

$$X_t \doteq (S_t, A_t, S_{t+1}, R_{t+1}, \phi_t, \rho_t),$$

which is generated according to a Markov process. Assume X_t is uniformly bounded for all t. Then (a) $(M_t, \mathcal{F}_t)_{t\geq 1}$ are martingale difference sequences, where $\mathcal{F}_t \doteq \sigma(Z_0, X_1, ..., X_t)$, where X_t sequence are increasing σ-fields ; (b) $\exists K_0$ s.t. $\mathbb{E}\left[M_{t+1}^2 \mid \mathcal{F}_t\right] \leq K_0(1 + \|Z_t\|^2)$.

Proof Let $G = \lim_{t\to+\infty} \mathbb{E}[G(X_t)]$ and $g = \lim_{t\to+\infty} \mathbb{E}[g(X_t)]$. Then there exists scalars K and $q > 0$ s.t.: (i) $\| \sum_{t=0}^{\infty} \mathbb{E}[G(X_t) - G \mid X_0 = x] \| \leq K(1+\|x\|^q)$; and (ii) $\| \sum_{t=0}^{\infty} \mathbb{E}[g(X_t) - g \mid X_0 = x] \| \leq K(1 + \|x\|^q)$.

Here, the sequence $\{\varrho_0, (X_t)_{t\geq 0}\}$ is defined on a probability space $(w, \mathcal{E}, \mathcal{P})$ and the sequence $\varrho_0, X_1, ..., X_t$ generates σ-field of events denoted by $\mathcal{F}_t \doteq \sigma(\varrho_0, X_0, \ldots, X_t)$. Let $H(\varrho_t, X_t) \doteq G_{t+1}\varrho_t + g_{t+1}$ and $h(\varrho) \doteq \mathbb{E}[H(\varrho, X_t)]$. For the case (i), we need to verify the following conditions in **??** the following conditions

Let $\nu(x, \varrho) = \sum_{t=0}^{\infty} \mathbb{E}[H(X_t, \varrho) - h(\varrho) \mid X_0 = x]$.

There exist C_i, q_i, $i = 1, ..., 3$ and $C_q (q > 0)$, such that for all $\varrho \in \mathbb{R}^{2d}$, then: (a) $\|H(\varrho, x)\| \leq C_1(1 + \|\varrho\|)(1 + \|x\|^{q_1})$; (b) $\mathbb{E}[\|X_t\|^q \mid X_0 = x] \leq C_q(1 + \|x\|^q)$; (c) $\|\nu(x, \varrho)\| \leq C_2(1 + \|\varrho\|)(1 + \|x\|^{q2})$; (d) $\|\nu(x, \varrho) - \nu(x, \varrho')\| \leq C_3\|\varrho - \varrho'\| \left(1 + \|x\|^{q3}\right)$. ■

Our main convergence theorem should be something like:

Theorem 3 (Convergence of linear GTD(0)) *Consider the linear GTD(0) iterations* (54a)-(54b). *For any finite MDP, for any uniformely bounded feature vectors $\phi : \mathcal{S} \mapsto [-M, M]^n$, and importance sampling ratio $\rho : \mathcal{S} \times \mathcal{A} \mapsto [0, M]$, there exists a positive constant $\eta_{\min}$, such that, for any positive step-size parameters α_t and β_t with the following conditions: (i) $\beta_t = \eta\alpha_t$, where $\eta > \eta_{\min}$, (ii) $\sum_{t=0}^{\infty} \alpha_t = +\infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < +\infty$. Then, J_{PBE} converges to zero with probability one. Moreover, $\eta_{\min}$ is the larger of 0 and the largest eigenvalue of the $n \times n$ symmetric matrix*

$$-C^{-1}\frac{A + A^\top}{2},$$

where C and A are defined by (22) *and* (24).

[GTD(0). Markov-noise theorem building on GDP theorem. On-policy GTD empirical experiments. GQ(0). this section also deals with existing solution methods, their failings, and the failure of other ideas. e.g., freaky features and averagers.]

To derive the GTD(0) algorithm we conduct gradient descent in the projected Bellman-error objective function (**??**). First, for simplicity let us define $\phi \doteq \phi_t$, $\phi' \doteq \phi_{t+1}$, $\rho \doteq \rho_t$, and using the identity $\mathbb{E}[\rho\phi\phi] = \mathbb{E}[\phi\phi]$, we have

$$
\begin{aligned}
&-\frac{1}{2}\nabla J_{\mathrm{PBE}}(\theta) \\
&= \mathbb{E}\Big[\rho(\phi-\gamma\phi')\phi^\top\Big]\,\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi] \\
&= \Big(\mathbb{E}\Big[\phi\phi^\top\Big]-\gamma\mathbb{E}\Big[\rho\phi'\phi^\top\Big]\Big)\,\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi] \\
&= \mathbb{E}[\rho\delta\phi]-\gamma\mathbb{E}\Big[\rho\phi'\phi^\top\Big]\,\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi] \\
&= \mathbb{E}[\rho\delta\phi]-\gamma\mathbb{E}\Big[\rho\phi'\phi^\top\Big]\,w_\pi(\theta), \qquad (59)
\end{aligned}
$$

where

$$
w_\pi(\theta)=\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi]\,, \qquad (60)
$$

which looks the same as the solution we get from least-mean-square (LMS) method in supervised learning by replacing $\rho\delta$ with supervised signals. Now assuming that we have an estimate for $w(\theta_t)$ at time step t, say w_t, we can sample from the gradient descent direction, resulting in the following $O(n)$ algorithm, called GTD(0):

$$
\begin{aligned}
\theta_{t+1} &= \theta_t+\alpha_t\rho_t\Big[\delta_t\phi_t-\gamma\phi_{t+1}(\phi_t^\top w_t)\Big]\,, \qquad (61)\\
& \qquad (62)
\end{aligned}
$$

Given that θ_t does not change quickly, we could use the supervised learning update $w_{t+1}=w_t+\beta_t(\rho_t\delta_t-\phi_t^\top w_t)\phi_t$, because given a fixed value of θ, the solution of this update is $w(\theta)$ as $\beta_t\to 0$. However, one also could use $w_{t+1}=w_t+\beta_t\rho_t(\delta_t-\phi_t^\top w_t)\phi_t$ update, because $\mathbb{E}\big[\rho_t\phi_t\phi_t^\top\big]=\mathbb{E}\big[\phi_t\phi_t^\top\big]$. This update is more sensible because when $\rho_t=0$ we do not expect to update the weights. Thus, we can use the following update for the w weights:

$$
w_{t+1} = w_t+\beta_t\rho_t(\delta_t-\phi_t^\top w_t)\phi_t. \qquad (63)
$$

Note that the update to θ_t is the sum of two terms: the first term is exactly the same as the TD(0) update (**??**), and the second term is essentially an adjustment or correction of the TD(0) update so that it follows the gradient of the PBE objective function. If the second parameter vector is initialized to $w_0=0$, and β_t is small, then this algorithm will start out making almost the same updates as conventional linear TD. Note also that after the convergence of θ_t, w_t will converge to zero again.

6.1 Empirical results

To begin to assess the practical utility of the new family of TD algorithms based on gradient-descent on on-policy problems, we compared the empirical learning rate of the following algorithms: 1) The algorithm by Sutton, Szepesvári and Maei (2008), which we call it here GTD1[2], 2) GTD2 in Sutton et al. (2009), 3) GTD(0), which is analogous to TDC in Sutton

2. Note this algorithm originally was called GTD(0). However, later more efficient algorithms were developed, thus, we call it here GTD1.

et al. (2009), and 4) conventional TD(0). Particularly, we compare the performance of these algorithms with linear function approximation on four small problems—three random-walk problems and a Boyan-chain problem. All of these problems were episodic, undiscounted, and involved only on-policy training with a fixed policy.

The random-walk problems were all based on the standard Markov chain (Sutton and Barto, 1998) with a linear arrangement of five states plus two absorbing terminal states at each end. Episodes began in the center state of the five, then transitioned randomly with equal probability to a neighbouring state until a terminal state was reached. The rewards were zero everywhere except on transition into the right terminal state, upon which the reward was +1.

We used three versions of this problem, differing only in their feature representations. The first representation, which we call *tabular features*, was the familiar table-lookup case in which, for example, the second state was represented by the vector $\phi_2 = (0, 1, 0, 0, 0)^\top$. The second representation, which we call *inverted features*, was chosen to cause extensive inappropriate generalization between states; it represented the second state by $\phi_2 = (\frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})^\top$ (the value $\frac{1}{2}$ was chosen to give the feature vectors unit norm). The third representation, which we called *dependent features*, used only $n = 3$ features and was not sufficient to solve the problem exactly. The feature vectors for the five states, left to right, were $\phi_1 = (1, 0, 0)^\top$, $\phi_2 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^\top$, $\phi_3 = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})^\top$, $\phi_4 = (0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^\top$, and $\phi_5 = (0, 0, 1)^\top$.

The Boyan-chain problem is a standard episodic task for comparing TD-style algorithms with linear function approximation (see Boyan, 2002 for details). We used the version with 14 states and $d = 4$ features.

We applied GTD(0), GTD2, GTD1, and TD(0) to these problems with a range of constant values for their step-size parameters. The parameter α was varied over a wide range of values, in powers of 2. For the GTD(0), GTD2, and GTD1 algorithms, the ratio $\eta = \beta/\alpha$ took values from the set $\{\frac{1}{4}, \frac{1}{2}, 1, 2\}$ for the random-walk problems; one lower power of two was added for the Boyan-chain problem. The initial parameter vectors, θ_0 and w_0, were set to 0 for all algorithms.

Each algorithm and parameter setting was run for 100-500 episodes depending on the problem, with the square root of the MSPBE, MSBE, NEU, and MSE (see Sutton et al. , 2009) computed after each episode, then averaged over 100 independent runs. Figure 5 summarizes all the results on the small problems using the MSPBE as the performance measure. The results for the other objective functions were similar in all cases and produced the same rankings. The standard errors are all very small (in the order of 10^{-2} to 10^{-3}), thus, are not shown. All the algorithms were similar in terms of their dependence and sensitivity to the step sizes. Overall, GTD1 learned the slowest, followed after a significant margin by GTD2, followed by GTD(0) and TD(0). It is important to note that our step-sizes in these empirical results are kept constant and as a result the RMSPBE as shown in Figure 5 (right sub-panel) will never go to zero. To get a measure of how well the new algorithms perform on a larger problem , such as 9x9 Computer Go (in on-policy domain), we refer the reader to Sutton et al. (2009). The results are remarkably consistent with what we saw in the small problems. The GTD1 algorithm was the slowest, followed by GTD2, GTD(0), and TD(0).

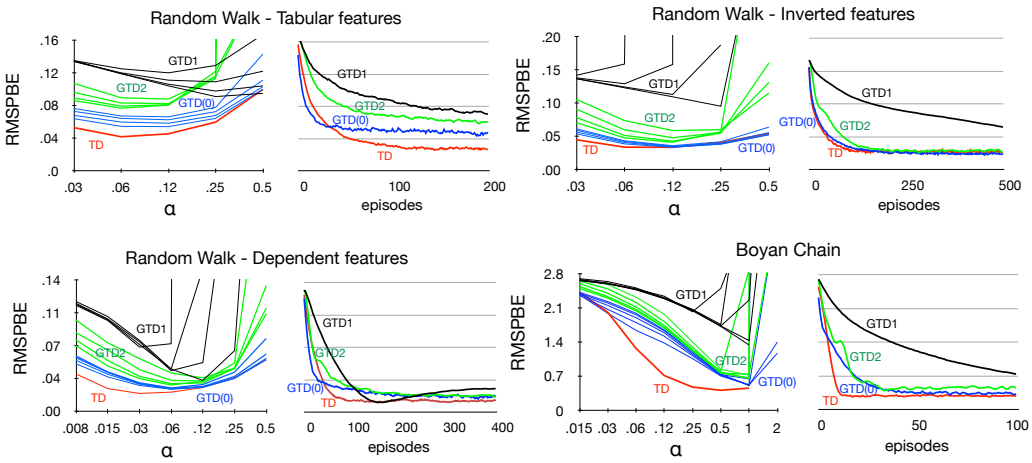


Figure 5: Empirical results on the four small problems—three versions of the 5-state random walk plus the 14-state Boyan chain. In each of the four panels, the right sub-panel shows a learning curve at best parameter values (RMSPBE denotes root of MSPBE objective function), and the left sub-panel shows a parameter study plotting the average height of the learning curve for each algorithm, for various $\eta = \beta/\alpha$, as a function of α. TD label shown in the graph represents TD(0) algorithm.

Finally, Figure 6 shows the results for an off-policy learning problem, demonstrating that the gradient methods converge on Baird's counterexample for which TD diverges.

6.2 GQ

Analogous to state-value functions, we define the λ-return for action-value functions:

$$G_t^\lambda(Q) \quad = \quad R_{t+1} + \gamma \Big[(1-\lambda) Q(S_{t+1}, A_{t+1}) + \lambda G_{t+1}^\lambda(Q) \Big], \tag{64}$$

where $Q(s,a)$ denotes the value of taking action a from state s, $\gamma \in (0,1]$, and $\lambda \in [0,1]$. Under MDP assumption, for every entry of state-action pair we get the following λ-weighted Bellman equation for action-value functions:

$$Q^\pi(s,a) \quad = \quad \mathbb{E}_\pi \Big[G_t^\lambda(Q^\pi) \mid S_t = s, A_t = a, \pi \Big] \doteq (T^{\pi\lambda} Q^\pi)(s,a),$$

where $T^{\pi\lambda}$ is a λ-weighted state–action version of the affine Bellman operator for the target policy π.

To estimate action-value functions, we use linear function approximation, where $Q \approx Q_\theta = \Phi\theta$ is the vector of approximate action values for each state–action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, and Φ is the matrix whose rows are the state–action feature vectors $\phi(s,a)^\top \in \mathbb{R}^d$.

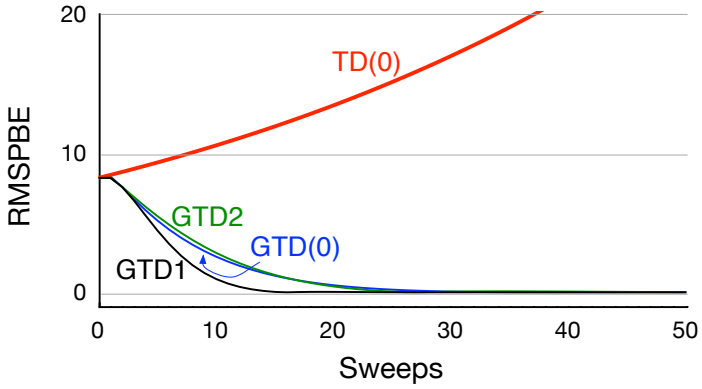


Figure 6: Learning curves on Baird's off-policy counterexample: TD diverges, whereas the gradient methods converge. This is the 7-state version of the "star" counterexample (Baird 1995), for which divergence is monotonic. Updating was done synchronously in dynamic-programming-like sweeps through the state space. For TD, $\alpha = 0.1$. For the gradient algorithms, $\alpha = 0.05$ and $\eta = 10$. The initial parameter value was $\theta_0 = (1, 1, 1, 1, 1, 1, 10, 1)^\top$, and $\gamma = 0.99$.

Following our approach for derivation of GTD(λ)in Section 8.4, we consider the following projected Bellman-error objective function:

$$J(\theta) = \|Q_\theta - \Pi T^{\pi\lambda} Q_\theta\|_d^2 = \left(\mathcal{P}_d^\pi \delta_t^\lambda(\theta)\phi_t\right)^\top \mathbb{E}[\phi_t \phi_t^\top]^{-1} \left(\mathcal{P}_d^\pi \delta_t^\lambda(\theta)\phi_t\right), \tag{65}$$

where δ_t^λ denotes λ-weighted TD error at time t:

$$\delta_t^\lambda(\theta) \doteq G_t^\lambda(\theta) - \theta^\top \phi_t, \tag{66}$$

and $G_t^\lambda(\theta)$ is

$$G_t^\lambda(\theta) \quad = \quad R_{t+1} + \gamma \left[(1-\lambda)\theta^\top \phi_{t+1} + \lambda G_{t+1}^\lambda(\theta)\right], \tag{67}$$

where $\phi_t \equiv \phi(S_t, A_t)$.

In the next section, first, we introduce the GQ(λ) algorithm, whose learning parameter is updated along the stochastic gradient descent of the above objective function, $J(\theta)$.

6.3 The GQ(λ) algorithm

First, we specify the GQ(λ) algorithm as follows: The weight vector $\theta \in \mathbb{R}^n$ is initialized arbitrarily. The secondary weight vector $w \in \mathbb{R}^n$ is initialized to zero. An auxiliary memory vector known as the eligibility trace $e \in \mathbb{R}^n$ is also initialized to zero. Their update rules are

$$\theta_{t+1} \quad = \quad \theta_t + \alpha_t \mathcal{I}_t \Big[\delta_t e_t - \gamma(1-\lambda)(w_t^\top e_t)\bar{\phi}_{t+1}\Big], \tag{68a}$$

$$w_{t+1} \quad = \quad w_t + \beta_t \mathcal{I}_t \left[\delta_t e_t - (w_t^\top \phi_t)\phi_t \right], \tag{68b}$$

where $\mathcal{I}_t$ is a desirable positive weight (see the GTD(λ) algorithm),

$$e_t = \phi_t + \gamma\lambda\rho_t e_{t-1}, \tag{69}$$

$$\delta_t = R_{t+1} + \gamma\theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t, \tag{70}$$

$$\bar{\phi}_t = \sum_a \pi(a \mid S_t)\phi(S_t, a), \tag{71}$$

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi(A_t \mid S_t)},$$

where ϕ_t is an alternate notation for $\phi(S_t, A_t)$, and $\alpha_t > 0$, $\beta_t > 0$, are positive step-size parameters for θ and w weights respectively.

In the next section we derive GQ(λ) based on gradient-descent in projected (λ-weighted) Bellman error objective function.

Algorithm 1 GQ(λ) with linear function approximation

Initialize w_0 to 0, and θ_0 arbitrarily.
Choose proper (small) positive values for α, β, and set values for $\gamma \in (0, 1]$, $\lambda \in [0, 1]$.
Repeat for each episode:
Initialize $e = 0$.
Take A_t from S_t according to π, and arrive at S_{t+1}.
Observe sample, (S_t, R_{t+1}, S_{t+1}) at time step t (with their corresponding state-action feature vectors).
for each observed sample **do**
    $\bar{\phi}_{t+1} \leftarrow \sum_a \pi(a \mid S_{t+1})\phi(S_{t+1}, a)$.
    $\delta_t \leftarrow R_{t+1} + \gamma\theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t$.
    $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi(A_t|S_t)}$.
    $e_t \leftarrow \phi_t + \rho_t\gamma\lambda e_{t-1}$.
    $\theta_{t+1} \leftarrow \theta_t + \alpha\mathcal{I}_t \left[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\bar{\phi}_{t+1}\right]$.
    $w_{t+1} \leftarrow w_t + \beta\mathcal{I}_t \left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
end for

6.4 Derivation of GQ(λ)

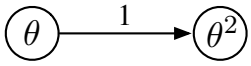
To derive GQ(λ), we follow the GTD(λ) derivation steps (for simplicity, first, we assume $\mathcal{I}_t$ functions are one).

7. Nonlinear function approximation

Remarks about challenges and open issues. Algorithms for all cases. Proof of convergence. Spiral counterexample.

The status of TDDP's limitation to linear function approximators is not completely clear. In practice, such algorithms have been widely used with nonlinear function approximators with good results. Tesauro's (1992, 1995) celebrated results with backgammon, for example, were obtained with a nonlinear neural-network function approximator. It is in fact extremely difficult to construct a example in which TDDP fails to converge under the on-policy distribution. The only such counterexample currently known is Tsitsiklis and Van Roy's spiral example, which is complex and contrived. We have tried to construct a simpler one without success. Moreover, we have recently shown that even, in the nonlinear case, all fixpoints of the TDDP update are stable—that if the approximator is started near a fixpoint it will converge to it (Maei, Sutton & Van Roy in preparation). It seems likely to us that there could be a significant further positive result to be obtained for nonlinear function approximators and TDDP. For the moment, however, there are no positive theoretical results for TDDP and nonlinear function approximators.

For linear function approximation, there always exists a θ at which the trip up and back leaves us in the same place and the PBE is zero. For nonlinear function approximators this is not true, as in this simple example:



The first state is assigned a value of θ (here a scalar) and the second state is assigned a value of θ^2 (thus making the approximator linear). For simplicity, assume d puts all weight on the first state. Then the value that minimizes the PBE (as well as the BE) is $\theta = \frac{1}{2\gamma}$, at which value the PBE (also the BE) is $\sqrt{3/4}$.

In this section, our goal is to extend the linear GTD(0) algorithm to nonlinear GTD(0). Particularly, we consider the case in which parametrized value function, V_θ, is an arbitrary differentiable nonlinear function. In the linear case, the objective function (MSPBE) was chosen as a projection of the Bellman error on a natural hyperplane - the subspace to which V_θ is restricted. However, in the nonlinear case, the value function is no longer restricted to a plane, but can move on a nonlinear surface.

More precisely, assuming that V_θ is a differentiable function of θ, $\mathcal{M} = \{V_\theta \in \mathbb{R}^{|S|} \,|\, \theta \in \mathbb{R}^n\}$ becomes a differentiable sub-manifold of $\mathbb{R}^{|S|}$. Projecting onto a nonlinear manifold is not computationally feasible; to get around this problem, we will assume that the parameter vector θ changes very little in one step (given that the value function is smooth and learning rates are usually small); in this case, the surface is locally close to linear, and we can project onto the tangent plane at the given point. We now detail this approach and show that this is indeed a good objective function.

7.1 Objective function for nonlinear function approximation

The *tangent plane* $P\mathcal{M}_\theta$ of $\mathcal{M}$ at θ is the hyperplane of $\mathbb{R}^{|S|}$ that *(i)* passes through V_θ and *(ii)* is orthogonal to the normal of $\mathcal{M}$ at θ. The *tangent space* $T\mathcal{M}_\theta$ is the translation of $P\mathcal{M}_\theta$ to the origin. Note that $T\mathcal{M}_\theta = \{\Phi_\theta a \,|\, a \in \mathbb{R}^n\}$, where $\Phi_\theta \in \mathbb{R}^{|S|\times n}$ is defined by $(\Phi_\theta)_{s,i} = \frac{\partial}{\partial \theta_i} V_\theta(s)$. Let Π_θ be the projection that projects vectors of $(\mathbb{R}^{|S|}, \|\cdot\|_d)$ to $T\mathcal{M}_\theta$. If $\Phi_\theta^\top D \Phi_\theta$ is non-singular then Π_θ can be written as

$$\Pi_\theta = \Phi_\theta (\Phi_\theta^\top D \Phi_\theta)^{-1} \Phi_\theta^\top D. \tag{72}$$

The objective function that we will optimize is:

$$J(\theta) \quad = \quad \|\Pi_\theta(T^\pi V_\theta - V_\theta)\| \,. \tag{73}$$

This is a natural generalization of the objective function defined by (**??**), as the plane on which we project is parallel to the tangent plane at θ. More precisely, let Υ_θ be the projection to $P\mathcal{M}_\theta$ and let Π_θ be the projection to $T\mathcal{M}_\theta$. Because the two hyperplanes are parallel, for any $V \in \mathbb{R}^{|S|}$, $\Upsilon_\theta V - V_\theta = \Pi_\theta(V - V_\theta)$. In other words, projecting onto the tangent space gives exactly the same distance as projecting onto the tangent plane, while being mathematically more convenient. Fig. 7 illustrates visually this objective function.

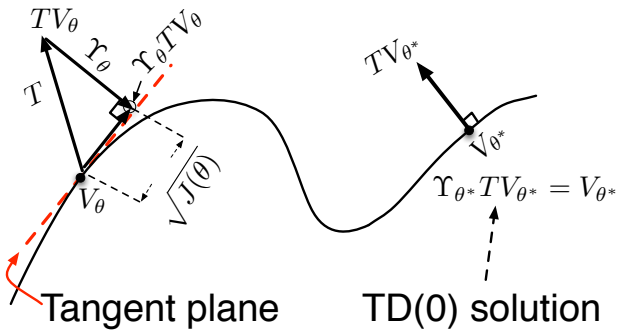


Figure 7: The MSPBE objective for nonlinear function approximation at two points in the value function space. Here T represents a Bellman operator. The figure shows a point, V_θ, at which, $J(\theta)$, is not 0 and a point, V_{θ^*}, where $J(\theta^*) = 0$, thus $\Upsilon_{\theta^*} T^\pi V_{\theta^*} = V_{\theta^*}$, so this is a TD(0) solution.

We now show that $J(\theta)$ can be re-written in the same way as done in (Sutton et al, 2009b).

Lemma 1 *Assume $V_\theta(s)$ is continuously differentiable as a function of θ, $\forall s \in \mathcal{S}$ s.t. $d(s) > 0$. Let $\delta(\theta) \doteq \delta(S, A, S'; \theta) = r(S, A, S') + \gamma\theta^\top\phi(S') - \theta^\top\phi(S)$ is a sample TD error for the sample transition (S, S') generated according to policy π, $\rho = \pi(A|S)/\pi(A|S)$, and $(S, \delta(\theta), \rho)$ be jointly distributed random variables, and assume that $\mathbb{E}[\nabla V_\theta(S)\nabla V_\theta(S)^\top]$ is nonsingular. Then*

$$J(\theta) = \mathbb{E}[\,\rho\;\delta(\theta)\,\nabla V_\theta(S)\,]^\top\,\mathbb{E}[\,\nabla V_\theta(S)\nabla V_\theta(S)^\top\,]^{-1}\,\mathbb{E}[\,\rho\;\delta\,\nabla V_\theta(S)\,]. \tag{74}$$

Proof *The derivation of the identity is similar to the derivation of mean-square projected Bellman objective function for off-policy formulation with linear function approximation (see Sec.**??**), except that here Π_θ is expressed by* (72). ■

Note that the assumption that $\mathbb{E}[\,\nabla V_\theta(S)\nabla V_\theta(S)^\top\,]^{-1}$ is non-singular is akin to the assumption that the feature vectors are independent in the linear function approximation case. We make this assumption here for convenience; it can be lifted, but the proofs become more involved.

Corollary 1 *Under the conditions of Lemma 1, $J(\theta) = 0$, if and only if V_θ is a TD(0) solution (i.e., if and only if it satisfies $\mathbb{E}[\,\rho\,\delta\,\nabla V_\theta(S)\,] = 0$).*

This is an important corollary (it is immediate), because it shows that optimizing the proposed objective function will indeed produce TD(0) solutions. We now proceed to compute the gradient of this objective.

Theorem 4 *Assume that* (i) *$V_\theta(s)$ is twice continuously differentiable in θ for any $s \in \mathcal{S}$ s.t. $d(s) > 0$ and* (ii) *$W(\cdot)$ defined by $W(\hat{\theta}) = \mathbb{E}[\nabla V_{\hat{\theta}}(S)\,\nabla V_{\hat{\theta}}(S)^\top]$ is non-singular in a small neighbourhood of θ. Let $\delta \doteq \delta(\theta)$, and (S, δ, ρ) be jointly distributed random variables. Let $\phi \equiv \nabla V_\theta(S)$, $\phi' \equiv \nabla V_\theta(S')$ and*

$$\hat{h}(\theta, u) = \mathbb{E}[\,\rho(\delta - \phi^\top u)\,\nabla^2 V_\theta(S) u\,], \tag{75}$$

where $u \in \mathbb{R}^n$. Then

$$-\frac{1}{2}\nabla J(\theta) = \mathbb{E}[\rho\delta\phi] - \gamma\mathbb{E}[\rho\phi'\phi^\top w(\theta)] - \hat{h}(\theta, w(\theta)), \tag{76}$$

where $w(\theta) = \mathbb{E}[\phi\,\phi^\top]^{-1}\,\mathbb{E}[\rho\delta\phi]$.

The main difference between Equation (76) and Equation (59), which shows the gradient for the linear case, is the appearance of the term $h(\theta, w)$, which involves second-order derivatives of V_θ (which are zero when V_θ is linear in θ).

Proof The conditions of Lemma 1 are satisfied, so (74) holds. Denote $\partial_i = \frac{\partial}{\partial\theta_i}$. From its definition and the assumptions, $W(u)$ is a symmetric, positive definite matrix, so $\frac{d}{du}(W^{-1})|_{u=\theta} = -\,W^{-1}(\theta)\,(\frac{d}{du}W|_{u=\theta})\,W^{-1}(\theta)$, where we use the assumption that $\frac{d}{du}W$ exists at θ and W^{-1} exists in a small neighborhood of θ. From this identity, we have:

$$\begin{aligned}
&-\frac{1}{2}\,[\nabla J(\theta)]_i \\
&= -(\partial_i\mathbb{E}[\delta\phi])^\top\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta\phi] - \frac{1}{2}\,\mathbb{E}[\delta\phi]^\top\,\partial_i\left(\mathbb{E}[\phi\phi^\top]^{-1}\right)\,\mathbb{E}[\delta\phi] \\
&= -(\partial_i\mathbb{E}[\rho\delta\phi])^\top\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi] + \frac{1}{2}\,\mathbb{E}[\rho\delta\phi]^\top\,\mathbb{E}[\phi\phi^\top]^{-1}(\partial_i\mathbb{E}[\phi\phi^\top])\,\mathbb{E}[\phi\phi^\top]^{-1}\,\mathbb{E}[\rho\delta\phi] \\
&= -\mathbb{E}[\partial_i(\rho\delta\phi)]^\top(\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi]) + \frac{1}{2}\,(\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi])^\top\,\mathbb{E}[\partial_i(\phi\phi^\top)]\,(\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi]).
\end{aligned}$$

The interchange between the gradient and expectation is possible here because of assumptions *(i)* and *(ii)* and the fact that $\mathcal{S}$ is finite. Now consider the identity

$$\frac{1}{2}x^\top\partial_i(\phi\phi^\top)x = \phi^\top x\;(\partial_i\phi^\top)x,$$

which holds for any vector $x \in \mathbb{R}^n$. Hence, using the definition of w,

$$\begin{aligned}
&-\frac{1}{2}\,[\nabla J(\theta)]_i \\
&= -\mathbb{E}[\partial_i(\rho\delta\phi)]^\top w + \frac{1}{2}w^\top\mathbb{E}[\partial_i(\phi\phi^\top)]w = -\mathbb{E}[\rho(\partial_i\delta)\phi^\top w] - \mathbb{E}[\delta(\partial_i\phi^\top)w] + \mathbb{E}[\phi^\top w(\partial_i\phi^\top)w].
\end{aligned}$$

Using $\nabla\delta = \gamma\phi' - \phi$ and $\nabla\phi^\top = \nabla^2 V_\theta(S)$, we get

$$\begin{aligned}
&-\frac{1}{2}\nabla J(\theta) \\
&= -\mathbb{E}[\rho(\gamma\phi' - \phi)\phi^\top w] - \mathbb{E}[(\rho\delta - \phi^\top w)\nabla^2 V(S)w] = \mathbb{E}[\rho(\phi - \gamma\phi')\phi^\top w] - \mathbb{E}[(\rho\delta - \phi^\top w)\nabla^2 V(S)w] \\
&= \mathbb{E}[\rho(\phi - \gamma\phi')\phi^\top w] - \mathbb{E}[\rho(\delta - \phi^\top w)\nabla^2 V(S)w].
\end{aligned}$$

Finally, observe that

$$\mathbb{E}[\rho(\phi - \gamma\phi')\phi^\top w] \;=\; \mathbb{E}[\rho(\phi - \gamma\phi')\phi]^\top \left(\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\rho\delta\phi]\right) = \mathbb{E}[\rho\delta\phi] - \mathbb{E}[\gamma\rho\phi'\phi^\top w],$$

where we have used $\mathbb{E}[\phi\phi^\top] = \mathbb{E}\big[\rho\phi\phi^\top\big]$, concluding the proof. ■

7.2 The nonlinear GTD(0) algorithm

Using the expression derived in Theorem 4, it suggests the following generalization of linear GTD(0), to the nonlinear case. Weight w_k is updated as before on the "faster" timescale:

$$w_{t+1} = w_t + \beta_k \rho_t (\delta_t - \phi_t^\top w_t)\phi_t. \tag{77}$$

The parameter vector θ_k is updated on a "slower" timescale, either according to

$$\theta_{t+1} = \Gamma\Big(\theta_t + \alpha_t\rho_t\Big\{\delta_t\phi_t - \gamma\phi_{t+1}(\phi_t^\top w_t) - h_t\Big\}\Big), \tag{78}$$

where

$$h_t = (\delta_t - \phi_t^\top w_t)\,\nabla^2 V_{\theta_t}(S_t)w_t \tag{79}$$

and $\Gamma : \mathbb{R}^n \to \mathbb{R}^n$ is a mapping that projects its argument into a set C, which is a parameter of the algorithm. Normally, one selects C to be a bounded set that has a smooth boundary and which is large enough so that the set of TD(0) solutions, $U = \{\,\theta \,|\, \mathbb{E}[\,\delta\,\nabla V_\theta(S)] = 0\,\}$, is subsumed by C. The purpose of this projection step is to prevent the algorithms' parameters diverge in the initial phase. Without the projection step this could happen due to the presence of the nonlinearities in the algorithm. Note that the projection is a common technique for stabilizing the transient behavior of stochastic approximation algorithms (Kushner and Yin, 1997). In practice, one selects C just large enough (by using *a priori* bounds on the size of the rewards and the derivative of the value function) in which case it is very likely that the parameter vector will not get projected at all during the execution of the algorithm. We also emphasize that the main reason for the projection is to facilitate convergence analysis. In many applications, this may not be needed at all.

Let us now analyze the computational complexity of these algorithms per update. Here we assume that we can compute $V_\theta(s)$ and its gradient with the cost of $\mathrm{O}(n)$ computation which is normally the case (e.g., neural networks). If the product of the Hessian of $V_\theta(s)$ and w can be computed in $O(n)$ time in (79), we immediately see that the computational cost of these algorithms per update will be $O(n)$. We show this is normally the case including neural networks. In the case of neural networks, let $V_\theta(s) = \sigma(\theta^\top x(s))$, where $\sigma(a) = \frac{1}{1+\exp(-a)}$, then $\nabla V_\theta(s) = [V_\theta(s)(1 - V_\theta(s))]\,x$ and $\nabla^2 V_\theta(s)w = \big[V_\theta(s)\,(1 - V_\theta(s))\,(1 - 2V_\theta(s))\,x^\top w\big]\,x$,

where the terms in [...] are scalar. The product of Hessian of $V_{\theta_k}(s)$ and w_k in (79) generally can be written as $\nabla(\nabla V_{\theta_k}(s)^\top w_k)$, because w_k does not depend on θ_k. As a result, because the scalar term, $\nabla V_{\theta_k}(s)^\top w_k$, cost only $\mathrm{O}(n)$ to compute its gradient which is a vector also has $\mathrm{O}(n)$ complexity. In general the observation that the product of a Hessian matrix and a vector of size n can be computed with the cost of $\mathrm{O}(n)$ is due to (Pearlmutter, 1994).

7.3 Convergence of Non-linear GTD(0) (two-time scales)

Let $\mathcal{C}(\mathbb{R}^n)$ be the space of $\mathbb{R}^n \to \mathbb{R}^n$ continuous functions. Define operator $\hat{\Gamma} : \mathcal{C}(\mathbb{R}^n) \to \mathcal{C}(\mathbb{R}^n)$ by

$$\hat{\Gamma} v\,(\theta) = \lim_{0<\varepsilon\to 0} \frac{\Gamma\big(\theta + \varepsilon\, v(\theta)\big) - \theta}{\varepsilon}.$$

In fact, because by assumption $\Gamma(\theta) = \arg\min_{\theta' \in C} \|\theta' - \theta\|$ and the boundary of $\mathcal{C}$ is smooth, $\hat{\Gamma}$ is well defined and in particular $\hat{\Gamma} v\,(\theta) = v(\theta)$ when $\theta \in \mathcal{C}^\circ$ and otherwise $\hat{\Gamma} v\,(\theta)$ is the projection of $v(\theta)$ to the tangent space of $\partial\mathcal{C}$ at $\Gamma(\theta)$. Consider the ODE

$$\dot{\theta} = \hat{\Gamma}(-\tfrac{1}{2}\nabla J)(\theta). \tag{80}$$

Let $\mathcal{K}$ be the set of all asymptotically stable fixed points of (80). By its definition, $\mathcal{K} \subset \mathcal{C}$. Further, for $\mathcal{U} \cap \mathcal{C} \subset \mathcal{K}$ (i.e., if θ is a TD(0)-solution that lies in $\mathcal{C}$ then it is an asymptotically stable fixed point of (80)).

The next theorem shows that under some technical conditions, the iterates produced by nonlinear GTD(0) converge to $\mathcal{K}$ with probability one. Thus, apart from the projection step, the algorithm converges to the stationary points of the objective function $J(\theta)$, which is the best result one can in general hope when using a stochastic gradient algorithm with a non-convex objective function.

Theorem 5 (Convergence of nonlinear GTD(0)) *Let $(S_t, R_{t+1}, S_{t+1})_{t\geq 0}$ is a stationary Markov process sequence generated according to the fixed policy π. Consider the nonlinear GTD(0) iterations* (77), (78)*. With positive deterministic step-size sequences that satisfy $\sum_{t=0}^{\infty} \alpha_t = \sum_{t=0}^{\infty} \beta_t = \infty$, $\sum_{t=0}^{\infty} \alpha_t^2$, $\sum_{t=0}^{\infty} \beta_t^2 < \infty$ and $\frac{\alpha_t}{\beta_t} \to 0$, as $t \to \infty$. Assume that for each $s \in \mathcal{S}$ such that $\mu(s) > 0$ (μ represents the state distribution), for all $\theta \in \mathcal{C}$, $V_\theta(s)$ is three times continuously differentiable. Further assume that for each $\theta \in \mathcal{C}$, $\mathbb{E}[\phi_\theta \phi_\theta^\top]$ is nonsingular. Then $\theta_k \to \mathcal{K}$, with probability one, as $t \to \infty$.*

Proof Just like the proof of convergence for linear GTD(0), for simplicity, let us use index k instead of time-step t and Let (S_k, R_k, S'_k) be a random transition whose law is the same as the law underlying $(S_t, R_{t+1}, S_{t+1})_{t\geq 0}$. Further, let $\phi_\theta = \nabla V_\theta(S)$, $\phi'_\theta = \nabla V_\theta(S')$, $\phi_k = \nabla V_{\theta_k}(S_k)$, and $\phi'_k = \nabla V_{\theta_k}(S'_k)$.

We begin by rewriting the updates (77)-(78) as follows:

$$w_{k+1} = w_k + \beta_k(f(\theta_k, w_k) + M_{k+1}), \tag{81}$$

$$\theta_{k+1} = \Gamma\big(\theta_k + \alpha_k(g(\theta_k, w_k) + N_{k+1})\big), \tag{82}$$

where

$$f(\theta_k, w_k) \;=\; \mathbb{E}[\rho_k \delta_k \phi_k | \theta_k] - \mathbb{E}[\rho_k \phi_k \phi_k^\top | \theta_k] w_k,$$

$$
\begin{aligned}
M_{k+1} &= \rho_k(\delta_k - \phi_k^\top w_k)\phi_k - f(\theta_k, w_k), \\
g(\theta_k, w_k) &= \mathbb{E}\Big[\rho_k\left(\delta_k\phi_k - \gamma\phi_k'\phi_k^\top w_k - h_k\right)|\theta_k, w_k\Big], \\
N_{k+1} &= \rho_k(\delta_k\phi_k - \gamma\phi_k'\phi_k^\top w_k - h_k) - g(\theta_k, w_k).
\end{aligned}
$$

We need to verify that there exists a compact set $\mathcal{B} \subset \mathbb{R}^{2n}$ such that (a) the functions $f(\theta, w)$, $g(\theta, w)$ are Lipschitz continuous over $\mathcal{B}$, (b) $(M_k, \mathcal{G}_k)$, $(N_k, \mathcal{G}_k)$, $k \geq 0$ are Markov noise sequences,

some conditions?

where $\mathcal{G}_k = \sigma(r_i, \theta_i, w_i, s_i, i \leq k; s_i', i < k)$, $k \geq 0$ are increasing σ-fields, (c) $\{(w_k(\theta), \theta)\}$ with $w_k(\theta)$ obtained as $\delta_k(\theta) = R_k + \gamma V_\theta(S_k') - V_\theta(S_k)$, $\phi_k(\theta) = \nabla V_\theta(S_k)$,

$$
w_{k+1}(\theta) = w_k(\theta) + \beta_k\rho_k\Big(\delta_k(\theta) - \phi_k(\theta)^\top w_k(\theta)\Big)\phi_k(\theta)
$$

almost surely stays in $\mathcal{B}$ for any choice of $(w_0(\theta), \theta) \in \mathcal{B}$, and (d) $\{(w, \theta_k)\}$ almost surely stays in $\mathcal{B}$ for any choice of $(w, \theta_0) \in \mathcal{B}$. From these, thanks to the conditions on the step-sizes, standard arguments (c.f.

we need to combine Benveniste and Borakar results. But because the iterates by assumption are bounded, is this easy?

) allow us to deduce that θ_k almost surely converges to the set of asymptotically stable fixed points of

$$
\dot{\theta} = \hat{\Gamma} F(\theta),
$$

where $F(\theta) = g(\theta, w_\theta)$. Here for $\theta \in \mathcal{C}$ fixed, w_θ is the (unique) equilibrium point of

$$
\dot{w} = \mathbb{E}[\rho\delta_\theta\phi_\theta] - \mathbb{E}[\phi_\theta\phi_\theta^\top]w, \tag{83}
$$

where $\delta_\theta = R + \gamma V_\theta(S') - V_\theta(S)$. Clearly, $w(\theta) = \mathbb{E}\big[\phi_\theta\phi_\theta^\top\big]^{-1}\mathbb{E}[\rho\delta_\theta\phi_\theta]$, which exists by assumption. Then by Theorem [add REF] it follows that $F(\theta) = -\frac{1}{2}\nabla J(\theta)$. Hence, the statement will follow once (a)–(d) are verified.

Note that (a) is satisfied because V_θ is three times continuously differentiable. For (b), we need to verify

some conditions on noise—we should add Benveniste conditions, but do we need to be strict here as the iterates are bounded?

Condition (c) follows since, by a standard argument (e.g., similar to the convergence of linear GTD(0)), $w_k(\theta)$ converges to $w(\theta)$, which by assumption stays bounded if θ comes from a bounded set. Now for condition (d), note that $\{\theta_k\}$ is uniformly bounded since $\theta_k \in C$, $\forall k$, and by assumption C is a compact set. ■

7.4 Empirical results

To illustrate the convergence properties of the algorithms, we applied them to the "spiral" counterexample mentioned in Section 12, originally used to show the divergence of TD(0) with nonlinear function approximation. The Markov chain with 3 states is shown in the

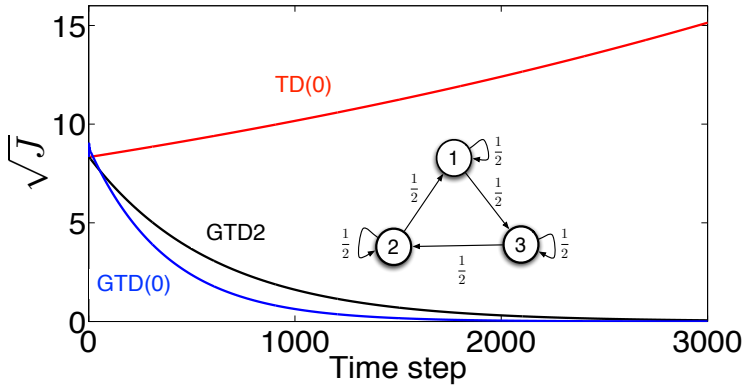


Figure 8: Empirical evaluation results for spiral counterexample. The selected parameters for GTD2 (see Maei et al., 2009), are $\alpha = 0.8$ and $\beta = 0.1$.

Figure 8. The reward is always zero and the discount factor is $\gamma = 0.9$. The value function has a single parameter, θ, and takes the nonlinear spiral form

$$V_\theta(s) = \Big(a(s)\cos\left(\hat{\lambda}\theta\right) - b(s)\sin\left(\hat{\lambda}\theta\right)\Big)\, e^{\epsilon\theta}.$$

The true value function is $V = (0, 0, 0)^\top$ which is achieved as $\theta \to -\infty$. Here we used $V_0 = (100, -70, -30)^\top$, $a = V_0$, $b = (23.094, -98.15, 75.056)^\top$, $\hat{\lambda} = 0.866$ and $\epsilon = 0.05$. Note that this is a degenerate example, in which our theorems do not apply, because the optimal parameter values are infinite. Hence, we run our algorithms without a projection step.

We also use constant learning rates, in order to facilitate gradient descent through an error surface which is essentially flat. For GTD(0) we used $\alpha = 0.5$, $\beta = 0.05$, For TD(0) we used $\alpha = 2 \times 10^{-3}$ (as argued by Tsitsiklis & Van Roy (1997), tuning the step-size does not help with the divergence problem). All step sizes are then normalized by $\|V_\theta^\top D \frac{d}{d\theta} V_\theta\|$.

Figure 8, shows the performance measure, $\sqrt{J}$, as a function of the number of updates (we used expected updates for all the algorithms). GTD(0) converge to the correct solution, while TD(0) diverges. We note that convergence happens despite the fact that this example is outside the scope of the theory.

The nonlinear GTD(0) has the same form as nonlinear TDC (Maei et al., 2009) on on-policy problems, and its performance has been assessed on 9x9 computer Go (see Maei, et al., 2009).

8. Multi-step gradient TDL

[to derive GTD(λ) and GQ(λ) we need to do forward-view/backward view and do derivation concisely because it share many parts of derivations for for GTD(0) and GQ(lambda). also we need to define] Before presenting the algorithms such as TD/RG, GTD and GQ, we remind our notation $\phi_t \doteq \phi(S_t)$.

8.1 Forward-view objective function based on importance weighting

Let us define the following $\lambda-$weighted return function:

$$G_t^{\lambda\rho}(\theta) = R_{t+1} + \gamma\left[(1-\lambda)\theta^\top \bar{\phi}_{t+1} + \lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta)\right],$$

where

$$\bar{\phi}_t \doteq \sum_a \pi(a \mid S_t)\phi(S_t, a), \quad \rho_t \doteq \frac{\pi(A_t \mid S_t)}{\pi(A_t \mid S_t)},$$

and let

$$\delta_t^{\lambda\rho}(\theta) \doteq G_t^{\lambda\rho}(\theta) - \theta^\top \phi_t. \tag{84}$$

The next Theorem shows $\mathcal{P}_d^\pi \delta_t^\lambda(\theta)\phi_t$ (see Eq. 108) can be replaced by $\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]$.

Theorem 6 *(Off-policy TD with important weighting) Let π and π denote the behavior and target policies, respectively. Consider $\delta_t^\lambda(\theta)$, $\delta_t^{\lambda\rho}(\theta)$ defined in equations* (109) (84)*. Then,*

$$\mathcal{P}_d^\pi \delta_t^\lambda(\theta)\phi_t = \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]. \tag{85}$$

Proof

We show this by expanding the right-hand side

$$\begin{aligned}
&\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma\Big((1-\lambda)\theta^\top \bar{\phi}_{t+1} + \lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta)\Big) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top \bar{\phi}_{t+1} \mid S_t = s, A_t = a, \pi\Big] \\
&\quad + \gamma\lambda\mathbb{E}\Big[\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top \bar{\phi}_{t+1} | S_t = s, A_t = a, \pi\Big] \\
&\quad + \sum_{s'} P(s' \mid s, a) \sum_{a'} \pi(a' \mid s') \frac{\pi(a'|s')}{\pi(a' \mid s')} \gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\rho}(\theta) \mid S_{t+1} = s', A_{t+1} = a'\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top \bar{\phi}_{t+1} \mid S_t = s, A_t = a, \pi\Big] \\
&\quad + \sum_{s',a'} P(s' \mid s, a)\pi(a' \mid s')\gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\rho} \mid S_{t+1} = s', A_{t+1} = a'\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top \bar{\phi}_{t+1} + \gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\rho}(\theta) \mid S_{t+1} = s', A_{t+1} = a'\Big] \mid S_t = s, A_t = a, \pi\Big],
\end{aligned}$$

which, as it continues to roll out, gives us $\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] = \mathbb{E}\big[G_t^\lambda(\theta) \mid S_t = s, A_t = a, \pi\big]$, and, eventually it yields $\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] = \mathcal{P}_d^\pi \delta_t^\lambda(\theta)\phi_t$, because the state-action distribution is based on behavior state-action pair distribution, d. ■

Therefore, from Theorem 6, the forward-view objective function (108) can be written as:

$$J(\theta) \;=\; \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]. \tag{86}$$

8.2 Backward-view objective function

In this section, we transform the TD forward-view to a mechanistic backward-view. To do this, we propose the following theorem, which provides a great tool for forward-view to backward-view transformation.

Theorem 7 *(Equivalence of TD forward-view and backward-view) The forward-view description of TD update is equivalence to the following mechanistic backward-view:*

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] = \mathbb{E}[\delta_t(\theta)e_t], \tag{87}$$

where $\delta_t^{\lambda\rho}(\theta)$ is defined in Equation (84), $\delta_t(\theta) = R_{t+1} + \gamma\theta^\top\bar{\phi}_{t+1} - \theta^\top\phi_t$, and e_t is eligibility trace vector at time-step t, and has the following recursive update:

$$e_t = \phi_t + \gamma\lambda\rho_t e_{t-1}. \tag{88}$$

Proof Consider

$$\begin{aligned}
G_t^{\lambda\rho}(\theta) &= R_{t+1} + \gamma\Big[(1-\lambda)\theta^\top\bar{\phi}_{t+1} + \lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta)\Big] \\
&= \Big[R_{t+1} + \gamma(1-\lambda)\theta^\top\bar{\phi}_{t+1}\Big] + \gamma\lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \\
&= \Big(R_{t+1} + \gamma\theta^\top\bar{\phi}_{t+1}\Big) - \gamma\lambda\theta^\top\bar{\phi}_{t+1} + \gamma\lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \\
&= \Big(R_{t+1} + \gamma\theta^\top\bar{\phi}_{t+1} - \theta^\top\phi_t + \theta^\top\phi_t\Big) - \gamma\lambda\theta^\top\bar{\phi}_{t+1} + \gamma\lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \\
&= \Big(\delta_t(\theta) + \theta^\top\phi_t\Big) - \gamma\lambda\theta^\top\bar{\phi}_{t+1} + \gamma\lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) + \gamma\lambda\rho_{t+1}\Big(\theta^\top\phi_{t+1} - \theta^\top\phi_{t+1}\Big) \\
&= \Big(\delta_t(\theta) + \theta^\top\phi_t\Big) + \gamma\lambda\rho_{t+1}\Big(G_{t+1}^{\lambda\rho}(\theta) - \theta^\top\phi_{t+1}\Big) + \gamma\lambda\Big(\rho_{t+1}\theta^\top\phi_{t+1} - \theta^\top\bar{\phi}_{t+1}\Big) \\
&= \Big(\delta_t(\theta) + \theta^\top\phi_t\Big) + \gamma\lambda\rho_{t+1}\delta_t^{\lambda\rho}(\theta) + \gamma\lambda\Big(\rho_{t+1}\theta^\top\phi_{t+1} - \theta^\top\bar{\phi}_{t+1}\Big),
\end{aligned}$$

thus,

$$\begin{aligned}
\delta_t^{\lambda\rho}(\theta) &= G_t^{\lambda\rho}(\theta) - \theta^\top\phi_t \\
&= \delta_t(\theta) + \gamma\lambda\rho_{t+1}\delta_t^{\lambda\rho}(\theta) + \gamma\lambda\Big(\rho_{t+1}\theta^\top\phi_{t+1} - \theta^\top\bar{\phi}_{t+1}\Big).
\end{aligned}$$

Note that the last part of the above equation has expected value of vector zero under the behavior policy because $\mathbb{E}[\rho_t\phi_t \mid S_t] = \sum_a \pi(a \mid S_t)\phi(S_t, a) \equiv \bar{\phi}_t$. Putting all these together, we can write the TD update (in expectation) in a simple way in terms of eligibility traces which leads to backward-view:

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}\phi_t\Big]$$

$$
\begin{aligned}
&= \mathbb{E}\Big[\Big(\delta_t + \gamma\lambda\rho_{t+1}\delta^{\lambda\rho}_{t+1}\Big)\,\phi_t\Big] + \mathbb{E}\Big[\gamma\lambda\theta^\top\left(\rho_{t+1}\phi_{t+1} - \bar{\phi}_{t+1}\right)\phi_t\Big] \\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}\Big[\gamma\lambda\rho_{t+1}\delta^{\lambda\rho}_{t+1}\phi_t\Big] + 0 \\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}\Big[\gamma\lambda\rho_t\delta^{\lambda\rho}_t\phi_{t-1}\Big] \\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}_b[\gamma\lambda\rho_t\Big(\delta_t + \gamma\lambda\rho_{t+1}\delta^{\lambda\rho}_{t+1} + \gamma\lambda\theta^\top\left(\rho_{t+1}\phi_{t+1} - \bar{\phi}_{t+1}\right)\Big)\phi_{t-1}] \\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}[\gamma\lambda\rho_t\delta_t\phi_{t-1}] + \mathbb{E}\Big[\gamma\lambda\rho_t\gamma\lambda\rho_{t+1}\delta^{\lambda\rho}_{t+1}\phi_{t-1}\Big] + 0 \\
&= \mathbb{E}[\delta_t\left(\phi_t + \gamma\lambda\rho_t\phi_{t-1}\right)] + \mathbb{E}\Big[\gamma\lambda_{t-1}\rho_{t-1}\gamma\lambda\rho_t\delta^{\lambda\rho}_t\phi_{t-2}\Big] \\
&\vdots \\
&= \mathbb{E}_b\Big[\delta_t\Big(\phi_t + \gamma\lambda\rho_t\phi_{t-1} + \gamma\lambda\rho_t\gamma\lambda_{t-1}\rho_{t-1}\phi_{t-2} + \cdots\Big)\Big] \\
&= \mathbb{E}[\delta_t e_t]\,, \qquad (89)
\end{aligned}
$$

where $e_t = \phi_t + \gamma\lambda\rho_t e_{t-1}$, which gives us a backward view algorithm for the TD(λ) update. ■

8.3 Stochastic gradient-descent derivation

Now from Equation (86) and Theorem 7, we get

$$J(\theta) = \mathbb{E}[\delta_t(\theta)e_t]^\top\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,.$$

Following the derivation of GTD(λ), we get:

$$-\frac{1}{2}\nabla J(\theta) \;=\; -\mathbb{E}\Big[\left(\gamma\bar{\phi}_{t+1} - \phi_t\right)e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,. \qquad (90)$$

We use the following identity:

$$
\begin{aligned}
\mathbb{E}\Big[\phi_t e_t^\top\Big] &= \mathbb{E}\Big[\phi_t(\phi_t + \gamma\lambda\rho_t e_{t-1})^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma\lambda\rho_t\phi_t e_{t-1}^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma\lambda\rho_{t+1}\phi_{t+1}e_t^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma\lambda\bar{\phi}_{t+1}e_t^\top\Big]\,,
\end{aligned}
$$

where we have used $\mathbb{E}\big[\rho_{t+1}\phi_{t+1}e_t^\top\big] = \mathbb{E}\big[\bar{\phi}_{t+1}e_t^\top\big]$.

Thus, from above and Equation (90), we get

$$
\begin{aligned}
&-\frac{1}{2}\nabla J(\theta) \\
&= -\mathbb{E}\Big[\left(\gamma\bar{\phi}_{t+1} - \phi_t\right)e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= -\mathbb{E}\Big[\gamma\bar{\phi}_{t+1}e_t^\top - \phi_t e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]
\end{aligned}
$$

$$
\begin{aligned}
&= -\mathbb{E}\Big[\gamma\bar{\phi}_{t+1}e_t^\top - \Big(\phi_t\phi_t^\top + \gamma\lambda\bar{\phi}_{t+1}e_t^\top\Big)\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= -\mathbb{E}\Big[\gamma(1-\lambda)\bar{\phi}_{t+1}e_t^\top - \phi_t\phi_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= \mathbb{E}[\delta_t(\theta)e_t] - \mathbb{E}\Big[\gamma(1-\lambda)\bar{\phi}_{t+1}e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= \mathbb{E}[\delta_t(\theta)e_t] - \mathbb{E}\Big[\gamma(1-\lambda)\bar{\phi}_{t+1}e_t^\top\Big]\,w(\theta), \qquad (91)
\end{aligned}
$$

where $w(\theta) = \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]$.

Thus, by direct sampling from the above gradient-descent direction and weight-duplication trick we get:

$$
\theta_{t+1} = \theta_t + \alpha_t\mathcal{I}_t\Big[\delta_t e_t - \gamma(1-\lambda)\bar{\phi}_{t+1}(e_t^\top w_t)\Big],
$$

$$
w_{t+1} = w_t + \beta_t\mathcal{I}_t\Big[\delta_t e_t - (w_t^\top\phi_t)\phi_t\Big],
$$

where positive $\mathcal{I}_t$ weighted are added based on our discussion on TD solution (see the GTD(λ) algorithm).

Convergence remarks: Convergence analysis of GQ(λ) is similar to GTD(0) (see the convergence remarks for GTD(λ)in Section 8.7).

The GTD(λ)**algorithm** By direct sampling from Equation ... and following we get the GTD(λ)algorithm :

$$
\theta_{t+1} = \alpha_t\Big[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\phi_{t+1}\Big], \qquad (92a)
$$

$$
w_{t+1} = \beta_t\Big[\delta_t e_t - (w_t^\top\phi_t)\phi_t\Big], \qquad (92b)
$$

where α_t, β_t are positive step-sizes at time-step t, δ_t refers to TD error at time t, $\rho_t = \frac{\pi(S_t,A_t)}{\mu(S_t,A_t)}$, and

$$
e_t = \rho_t\left(\phi_t + \gamma\lambda e_{t-1}\right),
$$

[In Algorithm 2, I have used $I_t > 0$, which represent a weight for the importance of state(s) at time t. I'm not sure, we should put it here.]

Eligibility traces are essential for TD learning because they bridge the temporal gaps in cause and effect when experience is processed at a temporally fine resolution. In previous sections we discussed one-step TD prediction. Several important properties of eligibility traces are as follows: 1) They make TD methods more like efficient incremental Monte-Carlo algorithms. For example, in TD(λ), $\lambda \in [0,1]$ refers to eligibility function and is equivalent to Monte-Carlo methods when $\lambda = 1$, 2) They are particularly of interest when reward is delayed by many steps, thus, by adjusting λ function we may get faster and efficient learning. For further detail description of eligibility traces we refer the reader to Sutton & Barto (1998).

Algorithm 2 GTD(λ)with linear function approximation

Initialize w_0 to 0, and θ_0 arbitrarily.
Choose proper (small) positive values for α, β, and set values for $\gamma \in (0, 1]$, $\lambda \in [0, 1]$.
Repeat for each episode:
Initialize $e = 0$.
Take A_t from S_t according to μ, and arrive at S_{t+1}.
Observe sample, $(\phi_t, R_{t+1}, \phi_{t+1})$ at time step t.
for each observed sample **do**
    $\delta_t \leftarrow R_{t+1} + \gamma\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t$.
    $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi(A_t|S_t)}$.
    $e_t \leftarrow \rho_t \left(\phi_t + \gamma\lambda e_{t-1}\right)$.
    $\theta_{t+1} \leftarrow \theta_t + \alpha\mathcal{I}_t \left[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\phi_{t+1}\right]$.
    $w_{t+1} \leftarrow w_t + \beta\mathcal{I}_t \left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
end for

8.4 Problem formulation and objectives

Without loss of generality we use linear value function approximation— similar analysis can be used for the nonlinear setting. We define the λ-return (function) according to

$$G_t^\lambda(V) \doteq R_{t+1} + \gamma\left[(1-\lambda)V(S_{t+1}) + \lambda G_{t+1}^\lambda\right],$$

where $\lambda \in [0, 1]$ is a constant eligibility trace parameter. For the table-look up case, a λ-weighted version of the Bellman equation follows from MDP property, which can be written as:

$$V^\pi(s) = \mathbb{E}\Big[G_t^\lambda(V^\pi) \mid S_t = s, \pi\Big] \doteq (T^{\pi\lambda}V^\pi)(s),$$

where $T^{\pi\lambda}$ is called the λ-weighted Bellman operator for policy π. For detailed description of λ-weighted Bellman equation we refer the reader to Tsitsiklis and Van Roy (1997).

Our objective is to find off-policy TD-solution, θ, which satisfies $V_\theta = \Pi T^{\pi\lambda}V_\theta$ (Π is defined in Equation (**??**)), while the data is generated according to a behavior policy π, with state distribution d, that is, $s \sim d(.)$.

Objective function We consider the following mean-square projected Bellman-error (MSPBE) objective function:

$$J(\theta) \quad = \quad \|V_\theta - \Pi T^{\pi\lambda}V_\theta\|_d^2. \tag{93}$$

Let us, first, consider the following definitions and identities. We start with the following definitions: $\phi_t \doteq \phi(S_t)$,

$$G_t^\lambda(\theta) \doteq R_{t+1} + \gamma\left[(1-\lambda)\theta^\top\phi_{t+1} + \lambda G_{t+1}^\lambda(\theta)\right], \tag{94}$$

$$\delta_t^\lambda(\theta) \doteq G_t^\lambda(\theta) - \theta^\top\phi_t, \quad \mathcal{P}_d^\pi\delta_t^\lambda(\theta)\phi_t \quad \doteq \quad \sum_s d(s)\mathbb{E}\Big[\delta_t^\lambda(\theta)|S_t = s, \pi\Big]\phi(s), \tag{95}$$

where $\mathcal{P}_d^\pi$ is an operator. And now consider the following identities:

$$\mathbb{E}\Big[\delta_t^\lambda(\theta)|S_t=s,\pi\Big]=(T^{\pi\lambda}V_\theta-V_\theta)(s),\quad \mathbb{E}\Big[\phi_t\phi_t^\top\Big]=\Phi^\top D\Phi,$$

$$\mathcal{P}_d^\pi\delta_t^\lambda(\theta)\phi_t \;=\; \Phi^\top D\left(T^{\pi\lambda}V_\theta-V_\theta\right),$$

where D and Φ are defined in Sec. **??**. Using the above definitions and identities and following, we have:

$$J(\theta)=\|V_\theta-\Pi T^{\pi\lambda}V_\theta\|_d^2=(\mathcal{P}_d^\pi\delta_t^\lambda(\theta)\phi_t)^\top\mathbb{E}[\phi_t\phi_t^\top]^{-1}(\mathcal{P}_d^\pi\delta_t^\lambda(\theta)\phi_t). \tag{96}$$

Practically, there are two major issues with the objective function (96): 1) The expectation term is with respect to the policy π, while the data is generated according to behavior policy π, 2) We cannot estimate forward-view TD error as it needs future data. We call the above objective function forward-view objective function.

To overcome the first issue, we use importance-weighting scenario (also see Sec **??**). In the next section, we show how to use importance weighting, and show a forward-view objective function whose expectation terms are with respect to behavior policy π. Later, we use the mechanistic TD backward-view to deal with the second issue.

8.5 Forward-view objective function

The expectation TD update term in $J(\theta)$, is with respect to target policy π. In order to convert it to π, we need to use the notion of importance sampling. After we conduct this step, we will show how to transform the forward-view TD update terms into mechanistic backward-view.

Importance-weighting: First, let us introduce the following recursive $\lambda-$return equation at time t,

$$G_t^{\lambda\rho}(\theta)=\rho_t\left(r_{t+1}+\gamma\left[(1-\lambda)\theta^\top\phi_{t+1}+\lambda G_{t+1}^{\lambda\rho}(\theta)\right]\right), \tag{97}$$

which is based on the likelihood ratio

$$\rho_t\doteq\frac{\pi(A_t\mid S_t)}{\pi(A_t\mid S_t)}. \tag{98}$$

Let

$$\delta_t^{\lambda\rho}(\theta) \;=\; G_t^{\lambda\rho}(\theta)-\theta^\top\phi_t, \tag{99}$$

then according to the following theorem we have $\mathcal{P}_d^\pi\delta_t^\lambda(\theta)\phi_t=\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]$.

Theorem 8 *(Off-policy TD with importance weighting) Let π and π denote the behavior and target policies, respectively. Consider $\delta_t^\lambda(\theta)$, $\delta_t^{\lambda\rho}(\theta)$ defined in equations* (95) (99). *Then,*

$$\mathcal{P}_d^\pi\delta_t^\lambda(\theta)\phi_t=\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]. \tag{100}$$

Proof

We show this by expanding the right-hand side

$$\begin{aligned}
\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s\Big] &= \mathbb{E}\Big[\rho_t \Big(R_{t+1} + \gamma(1-\lambda)\theta^\top \phi_{t+1}\Big) + \rho_t \gamma\lambda G_{t+1}^{\lambda\rho}(\theta) \mid S_t = s\Big] \\
&= \mathbb{E}\Big[\rho_t \Big(R_{t+1} + \gamma(1-\lambda)\theta^\top \phi_{t+1}\Big) \mid S_t = s\Big] + \rho_t\gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\rho}(\theta) \mid S_t = s\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top\phi_{t+1} \mid S_t = s, \pi\Big] \\
&\quad + \sum_{a,s'} P(s' \mid s, a)\pi(a \mid s)\frac{\pi(a|s)}{\pi(a \mid s)}\gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\gamma\rho}(\theta) \mid S_{t+1} = s'\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top\phi_{t+1} \mid S_t = s, \pi\Big] \\
&\quad + \sum_{a,s'} P(s' \mid s, a)\pi(a \mid s)\gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\gamma\rho}(\theta) \mid S_{t+1} = s'\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top\phi_{t+1} + \gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\gamma\rho}(\theta) \mid S_{t+1} = s'\Big] \mid S_t = s, \pi\Big],
\end{aligned}$$

which, as it continues to roll out, gives us $\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s\Big] = \mathbb{E}\big[G_t^{\lambda}(\theta) \mid S_t = s, \pi\big]$. And, eventually we get $\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] = \mathcal{P}_d^\pi \delta_t^\lambda(\theta)\phi_t$, because the state-distribution is based on behavior state-distribution, d. ■

Now using the above theorem, the forward-view objective function (96), can be written as

$$J(\theta) = \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]. \tag{101}$$

8.6 Backward-view objective function

The forward-view objective function is a function of $\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]$, which is not practical to estimate because it depends on future data. The following theorem, however, makes it possible through a mechanistic backward-view.

Theorem 9 *(Equivalence of the TD forward-view and backward-view) The forward-view description of TD update is equivalence to the following mechanistic backward-view:*

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] = \mathbb{E}[\delta_t(\theta)e_t], \tag{102}$$

where $\delta_t^{\lambda\rho}(\theta)$ is defined in Equation (99), $\delta_t(\theta)$ is the conventional TD error, $\delta_t(\theta) = R_{t+1} + \gamma\theta^\top\phi_{t+1} - \theta^\top\phi_t$, and e_t is the eligibility trace vector at time-step t, and has the following recursive update:

$$e_t = \rho_t\left(\phi_t + \gamma\lambda e_{t-1}\right). \tag{103}$$

Proof Consider

$$G_t^{\lambda\rho}(\theta) = \rho_t\Big(R_{t+1} + \gamma\Big[(1-\lambda)\theta^\top\phi_{t+1} + \lambda G_{t+1}^{\lambda\rho}(\theta)\Big]\Big)$$

$$
\begin{aligned}
&= \rho_t \left(R_{t+1} + \gamma \theta^\top \phi_{t+1} - \theta^\top \phi_t + \theta^\top \phi_t \right) - \rho_t \gamma \lambda \theta^\top \phi_{t+1} + \rho_t \gamma \lambda G_{t+1}^{\lambda\gamma\rho}(\theta) \\
&= \rho_t \left(R_{t+1} + \gamma \theta^\top \phi_{t+1} - \theta^\top \phi_t \right) + \rho_t \theta^\top \phi_t + \rho_t \gamma \lambda \left(G_{t+1}^{\lambda\gamma\rho}(\theta) - \theta^\top \phi_{t+1} \right) \\
&= \rho_t \delta_t(\theta) + \rho_t \theta^\top \phi_t + \rho_t \gamma \lambda \delta_{t+1}^{\lambda\gamma\rho}(\theta),
\end{aligned}
$$

thus,

$$
\begin{aligned}
\delta_t^{\lambda\rho}(\theta) &= G_t^{\lambda\rho}(\theta) - \theta^\top \phi_t \\
&= \rho_t \delta_t(\theta) + \rho_t \theta^\top \phi_t + \rho_t \gamma \lambda \delta_{t+1}^{\lambda\rho}(\theta) - \theta^\top \phi_t \\
&= \rho_t \delta_t(\theta) + (\rho_t - 1)\theta^\top \phi_t + \rho_t \gamma \lambda \delta_{t+1}^{\lambda\rho}(\theta).
\end{aligned}
$$

Also, consider the following identity:

$$
\begin{aligned}
\mathbb{E}\Big[(1-\rho_t)\theta^\top \phi_t \phi_t\Big] &= \sum_{s,a} d(s)\pi(a \mid s) \left(1 - \frac{\pi(a \mid s)}{\pi(a \mid s)}\right) \theta^\top \phi(s)\phi(s) \\
&= \sum_s d(s) \left(\sum_a \pi(a \mid s) - \sum_a \pi(a \mid s) \right) \theta^\top \phi(s)\phi(s) \\
&= \sum_s d(s) \left(1 - 1\right) \theta^\top \phi(s)\phi(s) \\
&= 0,
\end{aligned}
$$

and consequently, $\mathbb{E}\big[(1-\rho_t)\theta^\top \phi_t \phi_k\big] = 0$, for $k < t$.

Putting all above together, we get:

$$
\begin{aligned}
&\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] \\
&= \mathbb{E}\Big[\rho_t \delta_t(\theta)\phi_t + (\rho_t - 1)\theta^\top \phi_t + \rho_t \gamma \lambda \delta_{t+1}^{\lambda\rho}(\theta)\phi_t\Big] \\
&= \mathbb{E}[\rho_t \delta_t(\theta)\phi_t] + 0 + \mathbb{E}_\pi\Big[\rho_t \gamma \lambda \delta_{t+1}^{\lambda\rho}(\theta)\phi_t\Big] \\
&= \mathbb{E}\Big[\rho_t \delta_t(\theta)\phi_t + \rho_{t-1}\gamma\lambda \delta_t^{\lambda\rho}\phi_{t-1}\Big] \\
&= \mathbb{E}\Big[\delta_t{}^{\gamma\rho}(\theta)\phi_t + \rho_{t-1}\gamma\lambda \left(\rho_t \delta_t(\theta)\phi_t + (\rho_t - 1)\theta^\top \phi_t + \rho_t \gamma \lambda \delta_{t+1}^{\lambda\rho}(\theta)\phi_t\right)\phi_{t-1}\Big] \\
&= \mathbb{E}\Big[\rho_t \delta_t(\theta)\phi_t + \rho_{t-1}\gamma\lambda \left(\rho_t \delta_t(\theta) + \rho_t \gamma \lambda \delta_{t+1}^{\lambda\rho}(\theta)\right)\phi_{t-1}\Big] \\
&= \mathbb{E}\Big[\rho_t \delta_t(\theta)\left(\phi_t + \rho_{t-1}\gamma\lambda\phi_{t-1}\right) + \rho_{t-1}\gamma\lambda\rho_t\gamma\lambda\delta_{t+1}^{\lambda\rho}(\theta)\phi_{t-1}\Big] \\
&= \mathbb{E}\Big[\rho_t \delta_t(\theta)\left(\phi_t + \gamma\lambda\rho_{t-1}\phi_{t-1}\right) + \rho_{t-2}\gamma\lambda\rho_t\gamma\lambda\delta_t^{\lambda\rho}(\theta)\phi_{t-2}\Big] \\
&\;\;\vdots \\
&= \mathbb{E}[\delta_t(\theta)\rho_t\left(\phi_t + \rho_{t-1}\gamma\lambda\phi_{t-1} + \rho_{t-2}\gamma\lambda\rho_{t-1}\gamma\lambda\phi_{t-2} + \cdots\right)] \\
&= \mathbb{E}[\delta_t(\theta)e_t]\,, \qquad (104)
\end{aligned}
$$

where $e_t = \rho_t(\phi_t + \gamma\lambda e_{t-1})$. ■

8.7 Derivation of the GTD(λ) algorithm

Now from Equation (101) and Theorem 9, we get

$$J(\theta) = \mathbb{E}[\delta_t(\theta)e_t]^\top \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,.$$

Just like derivation of GTD(0), first, we compute the steepest-descent direction of $J(\theta)$ is:

$$\begin{aligned}
&-\frac{1}{2}\nabla J(\theta)\\
&= -\frac{1}{2}\nabla\Big(\mathbb{E}[\delta_t(\theta)e_t]^\top \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\Big)\\
&= -\nabla\mathbb{E}\Big[\delta_t(\theta)e_t^\top\Big] \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\\
&= -\mathbb{E}\Big[(\gamma\phi_{t+1} - \phi_t)\, e_t^\top\Big] \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,.
\end{aligned}$$

We use the following identity:

$$\begin{aligned}
\mathbb{E}\Big[\phi_t e_t^\top\Big] &= \mathbb{E}\Big[\phi_t\rho_t(\phi_t + \gamma\lambda e_{t-1})^\top\Big]\\
&= \mathbb{E}\Big[\phi_t\rho_t\phi_t^\top + \phi_t\gamma\lambda e_{t-1}^\top\Big]\\
&= \mathbb{E}\Big[\phi_t\rho_t\phi_t^\top + \phi_{t+1}\rho_t\gamma\lambda e_t^\top\Big]\\
&= \mathbb{E}\Big[\phi_t\phi_t^\top + \phi_{t+1}\gamma\lambda e_t^\top\Big]\,,
\end{aligned}$$

where we have used shifting indices trick and the following identities

$$\mathbb{E}\Big[\phi_t\rho_t\phi_t^\top\Big] = \mathbb{E}\Big[\phi_t\phi_t^\top\Big]\,, \quad \mathbb{E}\Big[\phi_{t+1}\rho_t\gamma\lambda e_t^\top\Big] = \mathbb{E}\Big[\phi_{t+1}\gamma\lambda e_t^\top\Big]\,.$$

Thus,

$$\begin{aligned}
-\mathbb{E}\Big[(\gamma\phi_{t+1} - \phi_t)\, e_t^\top\Big] &= \mathbb{E}\Big[\gamma\phi_{t+1}e_t^\top - \phi_t e_t^\top\Big]\\
&= -\mathbb{E}\Big[\gamma\phi_{t+1}e_t^\top - \Big(\phi_t\phi_t^\top + \phi_{t+1}\gamma\lambda e_t^\top\Big)\Big]\\
&= \mathbb{E}\Big[\phi_t\phi_t^\top - \gamma(1-\lambda)\phi_{t+1}e_t^\top\Big]\,.
\end{aligned}$$

Using the above identity, we get

$$\begin{aligned}
&-\frac{1}{2}\nabla J(\theta)\\
&= -\mathbb{E}\Big[(\gamma\phi_{t+1} - \phi_t)\, e_t^\top\Big] \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\\
&= \mathbb{E}\Big[\phi_t\phi_t^\top - \gamma(1-\lambda)\phi_{t+1}e_t^\top\Big] \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\\
&= \mathbb{E}[\delta_t(\theta)e_t] - \mathbb{E}\Big[\gamma(1-\lambda)\phi_{t+1}e_t^\top\Big] \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\\
&= \mathbb{E}[\delta_t(\theta)e_t] - \mathbb{E}\Big[\gamma(1-\lambda)\phi_{t+1}e_t^\top\Big]\, w(\theta), \qquad (105)
\end{aligned}$$

where $w(\theta) = \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]$.

The GTD(λ)algorithm By direct sampling from Equation (105) and following the GTD(0) derivations steps we get the GTD(λ)algorithm[3]:

$$\theta_{t+1} = \alpha_t \left[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\phi_{t+1}\right], \tag{106a}$$

$$w_{t+1} = \beta_t \left[\delta_t e_t - (w_t^\top \phi_t)\phi_t\right], \tag{106b}$$

where

$$\delta_t = r_{t+1} + \gamma_{t+1}\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t,$$

$$e_t = \rho_t \left(\phi_t + \gamma\lambda e_{t-1}\right),$$

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi(A_t \mid S_t)}.$$

The off-policy TD algorithm, GTD(0), that we have discussed so far always has been weighted by the agent behavior distribution. This weighting, although seems natural, but can change the quality of TD fixed point (see Kolter, 2011). However, by addicting a desired weighted function (as a function of data), GTD(0) can be easily generalized to the cases where it can cover variety of solution as has been discussed in Precupt et al. (2001) and Kolter (2011). Algorithm 3 shows how to do this by addicting a term $\mathcal{I}_t \in \mathbb{R}_+$ in the update, which can be some cumulative product of weights (Precupt, 2001) or some desirable state-occupancy weight based on history of observed data.

Algorithm 3 GTD(λ)with linear function approximation

Initialize w_0 to 0, and θ_0 arbitrarily.
Choose proper (small) positive values for α, β, and set values for $\gamma \in (0, 1]$, $\lambda \in [0, 1]$.
Repeat for each episode:
Initialize $e = 0$.
Take A_t from S_t according to π, and arrive at S_{t+1}.
Observe sample, $(\phi_t, R_{t+1}, \phi_{t+1})$ at time step t.
for each observed sample **do**
    $\delta_t \leftarrow R_{t+1} + \gamma\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t$.
    $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi(A_t|S_t)}$.
    $e_t \leftarrow \rho_t \left(\phi_t + \gamma\lambda e_{t-1}\right)$.
    $\theta_{t+1} \leftarrow \theta_t + \alpha\mathcal{I}_t \left[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\phi_{t+1}\right]$.
    $w_{t+1} \leftarrow w_t + \beta\mathcal{I}_t \left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
end for

Convergence remarks: Just like the convergence of GTD(0) in Section **??**, it can be shown that GTD(λ) is guaranteed to converge under standard conditions and proper choice of step-sizes. Please note addicting the positive weights ($\mathcal{I}_\sqcup$) does not change

3. Note, GTD(λ)is named after GTD(λ) in Maei (2011).

the convergence proof as the objective function for this case would be in the form of $J(\theta) = \mathbb{E}[\mathcal{I}_t \delta_t e_t]^\top \mathbb{E}\big[\mathcal{I}_t \phi_t \phi_t^\top\big]^{-1} \mathbb{E}[\mathcal{I}_t \delta_t e_t]$ (note $\mathcal{I}_\sqcup$ can be a function of generated data in the past, that is, history.).

8.8 GQ

Analogous to state-value functions, we define the λ-return for action-value functions:

$$G_t^\lambda(Q) \quad = \quad R_{t+1} + \gamma \Big[(1-\lambda) Q(S_{t+1}, A_{t+1}) + \lambda G_{t+1}^\lambda(Q)\Big], \tag{107}$$

where $Q(s,a)$ denotes the value of taking action a from state s, $\gamma \in (0,1]$, and $\lambda \in [0,1]$. Under MDP assumption, for every entry of state-action pair we get the following λ-weighted Bellman equation for action-value functions:

$$Q^\pi(s,a) \quad = \quad \mathbb{E}_\pi\Big[G_t^\lambda(Q^\pi) \mid S_t = s, A_t = a, \pi\Big] \doteq (T^{\pi\lambda} Q^\pi)(s,a),$$

where $T^{\pi\lambda}$ is a λ-weighted state–action version of the affine Bellman operator for the target policy π.

To estimate action-value functions, we use linear function approximation, where $Q \approx Q_\theta = \Phi\theta$ is the vector of approximate action values for each state–action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, and Φ is the matrix whose rows are the state–action feature vectors $\phi(s,a)^\top \in \mathbb{R}^d$.

Following our approach for derivation of GTD(λ)in Section 8.4, we consider the following projected Bellman-error objective function:

$$J(\theta) = \|Q_\theta - \Pi T^{\pi\lambda} Q_\theta\|_d^2 = \Big(\mathcal{P}_d^\pi \delta_t^\lambda(\theta)\phi_t\Big)^\top \mathbb{E}[\phi_t \phi_t^\top]^{-1} \Big(\mathcal{P}_d^\pi \delta_t^\lambda(\theta)\phi_t\Big), \tag{108}$$

where δ_t^λ denotes λ-weighted TD error at time t:

$$\delta_t^\lambda(\theta) \doteq G_t^\lambda(\theta) - \theta^\top \phi_t, \tag{109}$$

and $G_t^\lambda(\theta)$ is

$$G_t^\lambda(\theta) \quad = \quad R_{t+1} + \gamma \Big[(1-\lambda)\theta^\top \phi_{t+1} + \lambda G_{t+1}^\lambda(\theta)\Big], \tag{110}$$

where $\phi_t \equiv \phi(S_t, A_t)$.

In the next section, first, we introduce the GQ(λ) algorithm, whose learning parameter is updated along the stochastic gradient descent of the above objective function, $J(\theta)$.

8.9 The GQ(λ) algorithm

First, we specify the GQ(λ) algorithm as follows: The weight vector $\theta \in \mathbb{R}^n$ is initialized arbitrarily. The secondary weight vector $w \in \mathbb{R}^n$ is initialized to zero. An auxiliary memory vector known as the eligibility trace $e \in \mathbb{R}^n$ is also initialized to zero. Their update rules are

$$\theta_{t+1} \quad = \quad \theta_t + \alpha_t \mathcal{I}_t \Big[\delta_t e_t - \gamma(1-\lambda)(w_t^\top e_t)\bar{\phi}_{t+1}\Big], \tag{111a}$$

$$w_{t+1} \quad = \quad w_t + \beta_t \mathcal{I}_t \left[\delta_t e_t - (w_t^\top \phi_t)\phi_t \right], \tag{111b}$$

where $\mathcal{I}_t$ is a desirable positive weight (see the GTD(λ) algorithm),

$$e_t = \phi_t + \gamma\lambda\rho_t e_{t-1}, \tag{112}$$

$$\delta_t = R_{t+1} + \gamma\theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t, \tag{113}$$

$$\bar{\phi}_t = \sum_a \pi(a \mid S_t)\phi(S_t, a), \tag{114}$$

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi(A_t \mid S_t)},$$

where ϕ_t is an alternate notation for $\phi(S_t, A_t)$, and $\alpha_t > 0$, $\beta_t > 0$, are positive step-size parameters for θ and w weights respectively.

In the next section we derive GQ(λ) based on gradient-descent in projected (λ-weighted) Bellman error objective function.

Algorithm 4 GQ(λ) with linear function approximation

Initialize w_0 to 0, and θ_0 arbitrarily.
Choose proper (small) positive values for α, β, and set values for $\gamma \in (0, 1]$, $\lambda \in [0, 1]$.
Repeat for each episode:
Initialize $e = 0$.
Take A_t from S_t according to π, and arrive at S_{t+1}.
Observe sample, (S_t, R_{t+1}, S_{t+1}) at time step t (with their corresponding state-action feature vectors).
for each observed sample **do**
    $\bar{\phi}_{t+1} \leftarrow \sum_a \pi(a \mid S_{t+1})\phi(S_{t+1}, a)$.
    $\delta_t \leftarrow R_{t+1} + \gamma\theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t$.
    $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi(A_t|S_t)}$.
    $e_t \leftarrow \phi_t + \rho_t\gamma\lambda e_{t-1}$.
    $\theta_{t+1} \leftarrow \theta_t + \alpha\mathcal{I}_t \left[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\bar{\phi}_{t+1}\right]$.
    $w_{t+1} \leftarrow w_t + \beta\mathcal{I}_t \left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
end for

8.10 Derivation of GQ(λ)

To derive GQ(λ), we follow the GTD(λ) derivation steps (for simplicity, first, we assume $\mathcal{I}_t$ functions are one).

9. Focusing/views of off-policy TDL

10. Hybrid TDL

11. Conclusion

12. Popular counterexamples for TD(0) with function approximation

In this section we present some of the well known counterexamples that demonstrate the stability problem of TD(0) with function approximation.

Baird's Off-policy Counterexample Consider the 7-star version of the "star" counterexample (Baird, 1995; Sutton and Barto, 1998). The Markov decision process (MDP) is depicted in Fig. 3. The reward is zero in all transitions, thus the true value functions for any given policy is zero. The behavior policy, in this example, chooses the solid line action with probability of 1/7 and the dotted line action with probability of 6/7. The goal is to learn the value of a target policy that chooses the solid line more often than the probability of 1/7. In this example, the target policy choose the solid action with probability of 1.

Value functions are approximated linearly. Both TD(0) and dynamic programming (with incremental updates), however, will diverge on this example; that is, their learning parameters will go to $\pm\infty$ as is illustrated in Fig. 3.

Now, we turn into the stability issues of TD methods in conjunction with nonlinear function approximation. Despite linear TD(0), nonlinear TD(0) can become unstable and diverge even under on-policy training. The spiral counterexample, due to Tsitsiklis and Van Roy (1997), shows divergence of nonlinear TD(0) under on-policy training.

Spiral Counterexample (Tsitsiklis & Van Roy, 1997): Consider the Markov chain with 3 states as is shown in the left panel of Fig. 9. All state-state transitions are with probability of 1/2, the reward is always zero, and the discount factor is $\gamma = 0.9$. The parametrized (approximate) value function has a scalar parameter, θ, and takes the nonlinear spiral form

$$V_\theta(s) = \Big(a(s)\cos(\hat{\lambda}\theta) - b(s)\sin(\hat{\lambda}\theta)\Big)\, e^{\epsilon\theta}.$$

The true value function is $V = (0, 0, 0)^\top$, which is achieved as $\theta \to -\infty$. The right panel of the figure, demonstrates the value functions for each state; each axis corresponds to value function for a given state. As is illustrated in Fig. 9(c), the learning parameter θ diverges using nonlinear TD(0). Note, here we have used the expected TD(0) update in the plot to illustrate how the θ parameter evolves under TD learning.

Restricted features: Here, we provide our a counter-example with restricted features. The purpose of this counter-example is to demonstrate that convergence, most likely, can not be guaranteed with restricted features; that is, one can find a target policy for which the learning parameters can divergence. Fig. 10a, shows a very simple example that TD(0) diverges. One can think that by restricting the features of the next states one can guarantee the convergence of TD(0). In Fig. 10b, the norm of each feature as well as the size of feature vectors for the next states are less (or equal) than the starting states. However, TD(0) still diverges for this example. Similar counterexamples can be found with binary features with restricted next-state sizes.

Binary features with equal norm: Just like tabular features, one might think, binary features with the same norm possibly would guarantee the convergence of TD learning. Fig. 11 demonstrates an example that refutes this idea. If we write the expected TD(0)

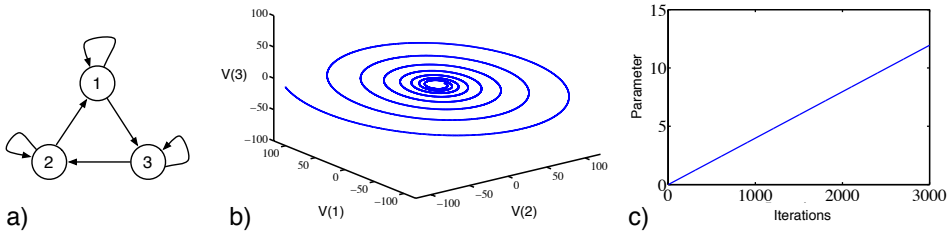


Figure 9: Spiral counterexample, which shows the divergence of nonlinear TD(0) under on-policy training. Panel a) shows the MDP problem with transitions and panel b) shows how the value functions evolve. Panel c) shows the learning parameter diverges to infinity. The parameter is updated according to expected TD(0) update (approximate dynamic programming style). Here, we used $V_0 = (100, -70, -30)^\top$, $a = V_0$, $b = (23.094, -98.15, 75.056)^\top$, $\hat{\lambda} = 0.866$ and $\epsilon = 0.05$.

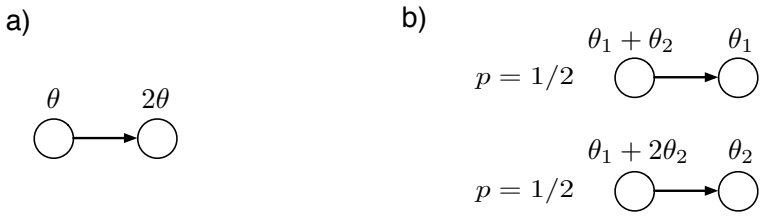


Figure 10: Restricted features. Panel a) shows a simple example for which TD(0) diverges. Here, the agent experience only one sample transition at every time step and the reward is zero. The value functions are shown on the top of each state (features are 1 and 2, respectively). Panel b) shows an example with restricted features and each sample transition is observed with probability of 1/2. Reward is zero for all these examples.

update in the form of $\mathbb{E}[\delta(\theta)\phi] = -A\theta + b$, then the real part eigenvalues of matrix A are -0.5 $+2$, $+2.75$ and $+2.75$. Because one of the real part eigenvalues is negative then it makes the linear system unstable and does the TD(0) iterate diverges.

13. Several Proposed Approaches for Solving the Stability Problem of TD Learning

Several approaches to the stability problem in reinforcement learning have been proposed, but none have been satisfactory in many ways; they do not satisfy our four desirable algorithmic features. (Also, as we mentioned, there has been several ideas for constraining

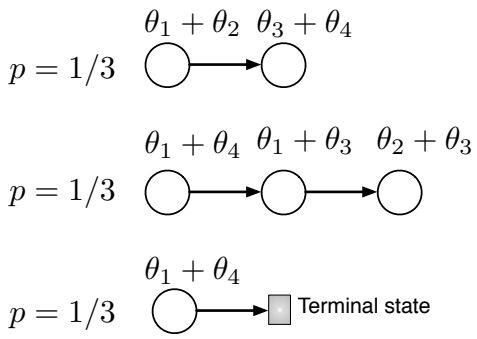


Figure 11: Binary features with equal norms. The value function of each state is shown with respect to its learning parameters. Here, we consider binary feature vectors with have equal norm. Just like example demonstrated in Fig. 10, TD(0) diverges for this example.

features, such as choosing binary feature vectors for each state with equal norm, and one always can find a counterexample for such forms of feature construction.)

One idea for retaining all four desirable features is to use cumulative products of target-to-behavior-policy likelihood ratios to re-weight TD updates. In principle it uses cumulative products of importance-weightings (likelihood ratios) over the data trajectory. In other words, at any given time t, the TD update is re-weighted by cumulative products of importance-weighting ratios up to time t, so that its expected value is in the same direction as on-policy update according to the target policy (Precup et al., 2000; 2001). Convergence can sometimes then be assured by existing results on the convergence of on-policy methods (Tsitsiklis and Van Roy 97; Tadić 2001). However, the importance-weighting weights are cumulative products of many target-to-behavior-policy likelihood ratios, and consequently they, and the corresponding updates, may be of very high variance.

The use of "recognizers" to construct the target policy directly from the behavior policy (Precup et al., 2006) is one strategy for limiting the variance; another is careful choice of the target policies (Precup et al., 2001). However, it remains the case that for all of such methods to date there are always choices of problem, behavior policy, and target policy for which the variance is infinite, and thus for which there is no guarantee of convergence.

The residual gradient (RG) method (Baird, 1995) has also been proposed as a way of obtaining all four desirable features. However, as we explained in Section **??** it has some fundamental drawbacks.

Gordon (1995) and others have questioned the need for linear function approximation. He has proposed replacing linear function approximation with a more restricted class of approximation, known as *averagers*, that never extrapolate outside the range of the observed data and thus cannot diverge (see also Szepesvari & Smart, 2004). Rightly or wrongly, averagers have been seen as being too constraining and have not been used on large applications involving online learning. Linear methods, on the other hand, have been widely used (e.g. Baxter, Tridgell & Weaver, 1998; Schaeffer, Hlynka & Jussila, 2001).

Linear function approximation is most powerful when very large numbers of features are used, perhaps millions of features (e.g., as in Silver et al., 2007). In such cases, methods with $O(n)$ complexity are required.

The stability problem of TD methods in conjunction with nonlinear function approximation is more severe, and nonlinear TD methods can diverge even for the case of on-policy learning (Tsitsiklis and Van Roy, 1997). There has been little work on addressing this stability problem, and proposed methods either are restricted to particular conditions or only partially solve the problem. For example, the Bridge algorithm by Papavassiliou and Russell (1999) uses a variant of TD learning and is convergent with nonlinear function approximation. However, it is a complex algorithm with high computational complexity, which hampers its practicality and also it does not have all of our four desirable algorithmic features.

14. General Value Functions

In this section we extend GQ(λ) and GTD(λ), which we developed in the previous sections, to a more general settings, including general value functions (GVFs), terminal-reward functions (outcomes), and allow policies to terminate at any given state with a termination (probability) function. The GVFs are introduced in Sutton et al. (2011) (also see Maei & Sutton, 2010; Maei, 2011).

In standard RL, the most common type of prediction is the expected total or discounted future reward, while following a policy. However, rewards could also be represented in the form of transient signals while acting—transients are a measure of what happens during the trajectory rather than its end. We denote the transient signal $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (note, we will show random variable with capital letters).

The second type of prediction is the expected outcome of policy upon its termination. We call this function the *outcome target function*, or *terminal-reward function*, $z : \mathcal{S} \to \mathbb{R}$, where $z(s)$ is the outcome that the agent receives if the policy terminates at state s.

Finally, the prediction could conceivably be a mixture of both a transient *and* an outcome. Here we will present the algorithm for predictions with both an outcome part z and a transient part r, with the two added together. In the common place where one wants only one of the two, the other is set to zero.

Now we can start to state the goal of learning more precisely. In particular, we would like our prediction to be equal to the expected value of the outcome target function at termination plus the cumulative sum of the transient reward function along the way. Thus, conventional action-value functions are defined in the following general form

$$Q^\pi(s,a) \equiv \mathbb{E}[R_{t+1} + R_{t+2} + \cdots + R_{t+k} + Z_{t+k} \mid S_t = s, A_t = a, \pi, \gamma], \tag{115}$$

where $\gamma : \mathcal{S} \to [0,1]$ is a discount function representing the probability of continuing to evaluate policy π from a given state, $Q^\pi(s,a)$ denotes action value function that evaluates policy π given state-action pair s, a, and its termination probability $1-\gamma(s)$. We call these action-value functions, general value functions (GVFs).

Our definition of GVFs does not involve discounting factor, as policies will terminate after some finite time—due to termination probability function $1-\gamma$. Later we will see that

γ function can be interpreted as discounting factor—$\gamma(s)$ indicates probability of continuing policy, π, from state s.

The Equation (115) describes the value functions in a Monte Carlo sense, but of course we want to include the possibility of temporal-difference learning. To do this, first we re-write the Equation (115) in the following bootstrapping form, which is derived under MDP assumption:

$$
\begin{aligned}
& Q^\pi(s,a) \\
&= \mathbb{E}[R_{t+1} + R_{t+2} + \cdots + R_{t+k} + z_{t+k} \mid S_t = s, A_t = a, \pi, \gamma] \\
&= \sum_{s'} P(s' \mid s,a) \left[r(s,a,s') + (1-\gamma(s'))z(s') + \gamma(s') \sum_{a'} \pi(a' \mid s') Q^\pi(s',a') \right] \\
&\doteq (T^\pi Q)(s,a), \qquad (116) \\
&\approx Q_\theta(s,a) = \theta^\top \phi(s,a) \qquad (117)
\end{aligned}
$$

where T^π has the form of Bellman operator for any given state-action pair, thus, we call the above equation a Bellman equation.

In the next section we show how to learn the parameter vector θ through a gradient TD method.

14.1 GQ(λ) for Learning GVFs

Table (5) shows how to use GQ(λ) with linear function approximation for GVFs.

Algorithm 5 GQ(λ) for Learning GVFs

Initialize w_0 to 0, and θ_0 arbitrarily.
Choose proper (small) positive values for α, β.
Repeat for each episode:
Initialize $e = 0$.
Take A_t from S_t according to π, and arrive at S_{t+1}.
Observe sample, $(S_t, R_{t+1}, Z_{t+1}, S_{t+1})$ at time step t (with their corresponding state-action feature vectors).
for each observed sample **do**
  $\bar{\phi}_{t+1} \leftarrow \sum_a \pi(a \mid S_{t+1})\phi(S_{t+1}, a)$, and
  $\lambda_t \leftarrow \lambda(S_t)$, $\gamma_t \leftarrow \gamma(S_t)$,$\gamma_{t+1} \leftarrow \gamma(S_{t+1})$.
  $\delta_t \leftarrow \big(R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\theta_t^\top \bar{\phi}_{t+1}\big) - \theta_t^\top \phi_t$.
  $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi(A_t|S_t)}$.
  $e_t \leftarrow \phi_t + \rho_t\gamma_t\lambda_t e_{t-1}$.
  $\theta_{t+1} \leftarrow \theta_t + \alpha\mathcal{I}_t \big[\delta_t e_t - \gamma_{t+1}(1-\lambda_{t+1})(e_t^\top w_t)\bar{\phi}_{t+1}\big]$.
  $w_{t+1} \leftarrow w_t + \beta\mathcal{I}_t \big[\delta_t e_t - (\phi_t^\top w_t)\phi_t\big]$.
end for

To derive GQ(λ) with linear function approximation for GVFs, we follow the derivation of GQ(λ) for standard value functions (see also Maei, 2011). GQ(λ) has been implemented

in Horde architecture (Sutton et al., 2011), in a robotic task, which involves a complex and non-stationary environment.

14.2 GTD(λ) for Learning GVFs

Analogous to action-value functions, general state-value functions can be learned through a gradient TD method for GVFs. The state-value function are needed for various optimal control methods, such as policy gradient method with actor-critic architecture for predicting grounded world-knowledge (Sutton, 2009).

Table (6) shows how to use GTD(λ) for GVFs with varying eligibility traces. For the detailed derivations see Maei (2011).

Algorithm 6 GTD(λ) for GVFs

Initialize w_0 to 0, and θ_0 arbitrarily.
Choose proper (small) positive values for α, β.
Repeat for each episode:
Initialize $e = 0$
Take A_t from S_t according to π, and arrive at S_{t+1}.
Observe sample, $(\phi_t, R_{t+1}, Z_{t+1}, \phi_{t+1})$, where $\phi_t = \phi(S_t)$.
for each observed sample **do**
  $\lambda_t \leftarrow \lambda(S_t)$, $\gamma_t \leftarrow \gamma(S_t)$,$\gamma_{t+1} \leftarrow \gamma(S_{t+1})$.
  $\delta_t \leftarrow \big(R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\theta_t^\top \phi_{t+1}\big) - \theta_t^\top \phi_t$.
  $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi(A_t|S_t)}$.
  $e_t \leftarrow \rho_t \left(\phi_t + \gamma_t \lambda_t e_{t-1}\right)$.
  $\theta_{t+1} \leftarrow \theta_t + \alpha \mathcal{I}_t \left[\delta_t e_t - \gamma_{t+1}(1-\lambda_{t+1})(e_t^\top w_t)\phi_{t+1}\right]$.
  $w_{t+1} \leftarrow w_t + \beta \mathcal{I}_t \left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
end for

15. Discussions

In this paper, we presented a new family of temporal-difference learning algorithms based on gradient descent. Our gradient-TD algorithms can be viewed as performing stochastic gradient-descent in a mean-square projected Bellman-error (PBE) objective function whose optimum is the TD-solution. Another key technique was introducing a set of auxiliary weights that were used in the update of actual parameters. The auxiliary weights were trained online, based on a stochastic update rule. All of our gradient-TD algorithms are guaranteed to converge.

The recent gradient TD methods (with TDC-style update, that is, TD update and a correction term), such as GTD(λ), seems more effective than GTD2 and GTD1 that were developed in Sutton, Szepesvári and Maei (2008) and Sutton et al. (2009). Such gradient TD algorithms, for the first time, have made it possible to have TD learning with linear complexity both in terms of memory and per-time-step computation while retaining their stability for general settings, including nonlinear function approximation and off-policy learning.

We also incorporated eligibility traces into our gradient-TD methods. Particularly, we considered a general scenario that eligibility trace function can be an arbitrary function of state, thus, can vary over time.

Another contribution is extending and making gradient-TD algorithms suitable for learning general value functions (GVFs). In standard RL, the value of a state is a measure of predicted future rewards from that state by following a policy. However, in the real-world, predictions are not necessarily in the form of future rewards. We would like to be able to answer questions such as: "If I keep moving forward now, would I bump to a wall after few second?" This question not only considers the outcome of moving forward after a few second, it also suggest that the policy of moving forward is only excited in particular states. The GVFs formulation is suitable for answering these temporally abstract predictions, which are also essential for representing abstract, higher level-knowledge about courses of action, or options. Here, we let policies to have their own activation and termination conditions, thus, making them suitable for option-conditional predictions (Sutton et al., 1998; 1999).

Although, in this paper, we focused on TD methods in the context of reinforcement learning, all these new ideas and methods can also be extended to various approximate DP methods. A good way to understand the impact of this work is to see it a a way of curing the curse-of-dimensionality.

The curse appears in large class of decision-making problems or problems that involve learning from interaction. Dynamic Programming (DP) is a general approach for solving complex decision-making problems. However, due to Bellman's curse-of-dimensionality it has not been extended in solving large-scale applications. The curse is considered as a malediction that has stopped scientists from finding exact solutions for large number of practical problems (Bellman, 1961, p. 94). Approximate DP methods—a collection of techniques for finding approximate DP solution— have been proposed to cure the curse.

However, approximate DP methods have cured the curse only partially. Baird's counterexample suggests we may have a stability problem when using approximate DP methods. Particularly, the problem arises when we are interested in algorithms that have the following desirable features: incremental, online, and linear complexity both in terms of memory and per-time-step computation.

The GQ(λ) can also be extended to control problems (Maei, et al. 2010, Maei, 2011). The future research is to conduct a through convergence analysis for control case.

References

Antos, A., Munos, R., and Szepesvári, Cs. (2007). Fitted Q-iteration in continuous action-space MDPs. In *Advances in Neural Information Processing Systems 20*, pp. 916. MIT Press.

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

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

Baird, L. C. (1999). Reinforcement Learning Through Gradient Descent. *PhD thesis*, Carnegie-Mellon University.

Bellman, R. (1961). *Adaptive control Processes*. Princeton University Press.

Benveniste, A., Metivier, M., Priouret, P. (1990). Adaptive Algorithms and Stochastic Approximations. Springer-Verlag.

Borkar, V. S. (1997). Stochastic approximation with two timescales. *Systems and Control Letters 29*:291-294.

Borkar, V. S. (2008). Stochastic Approximation: A Dynamical Systems Viewpoint. Cambridge University Press.

Borkar, V. S. and Meyn, S. P. (2000). The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control And Optimization 38*(2):447–469.

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

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

Crites, R. H. & Barto, A.G. (1995). Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, pp. 1017-1023. MIT Press.

Delyon, B. (1996). General results on the convergence of stochastic algorithms. *IEEE Trans. Automat. Contr., vol. 41*, pp. 1245–1255.

Kolter, J. Z. (2011). The Fixed Points of Off-Policy TD. In *Advances in Neural Information Processing Systems*.

Kushner, H. J. & Yin, G. G. (2003). Stochastic Approximation Algorithms and Applications. Second Edition, Springer-Verlag.

Geramifard, A., Bowling, M., Sutton, R. S. (2006). Incremental least-square temporal difference learning. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pp. 356–361.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 261268. Morgan Kaufmann.

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

Ljung, L. (1977). Analysis of recursive stochastic algorithms. *IEEE Tran. Automat. Control*, 22:551575, 1977.

Papavassiliou, V. A., and Russell, S. (1999). Convergence of reinforcement learning with general function approximators. In *Proceedings of the 16th international joint conference on Artificial intelligence*, pp. 748–757. Morgan Kaufmann.

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

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

Maei, H. R., Szepesvari, Cs, Bhatnagar, S., Precup, D., Silver D., Sutton, R. S. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems 22*. MIT Press.

Maei, H. R., Szepesvari, Cs, Bhatnagar, S., Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 719–726. Omnipress.

Pearlmutter, B. A (1994). Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160.

Precup, D., Sutton, R. S. and Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. *Proceedings of the 18th International Conference on Machine Learning*, pp. 417–424.

Precup, D., Sutton, R. S., Paduraru, C., Koop, A., Singh, S. (2006). Off-policy learning with recognizers. In *Advances in Neural Information Processing Systems 18*.

Precup, D., Sutton, R. S., Singh, S. (2000). Eligibility traces for off-policy policy evaluation. *Proceedings of the 17th International Conference on Machine Learning*, pp. 759–766. Morgan Kaufmann.

Schaeffer, J., Hlynka, M., Jussila, V. (2001). Temporal difference learning applied to a high-performance game-playing program. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 529534.

Scherrer, B. (2010). Should one compute the Temporal Difference fix point or minimize the Bellman Residual? The unified oblique projection view. In *In Proceedings of the 27th International Conference on Machine Learning*, pp. 959-966. Omnipress.

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

Sutton, R. S. (2009). The grand challenge of predictive empirical abstract knowledge. *Working Notes of the IJCAI-09 Workshop on Grand Challenges for Reasoning from Experiences*.

Sutton, R. S., Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., Precup D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence 112*:181–211.

Sutton, R. S., Precup D., and Singh, S. (1998). Intra-Option Learning about Temporally Abstract Actions. In *Proceedings of the 15th International Conference on Machine Learning*, pp. 556-564. Morgan Kaufmann.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsuper-

vised sensorimotor interaction. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems.*

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

Sutton, R. S., Maei, H. R, Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs. & Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 993–1000. Omnipress.

Szepesvári, Cs., Smart, W. D. (2004). Interpolation-based Q-learning. In *Proceedings of the 21th International Conference on Machine Learning*, pp .791–798. ACM.

Tadić, V. (2001). On the convergence of temporal-difference learning with linear function approximation. In *Machine Learning*, 42:241267.

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

Van Roy, B. (1999). Learning and Value Function Approximation in. Complex Decision Processes. *PhD thesis*, MIT.