True Online Temporal-Difference Learning

Harm van Seijen harm.vanseijen@ualberta.ca
A. Rupam Mahmood ashique@ualberta.ca
Patrick M. Pilarski patrick.pilarski@ualberta.ca
Marlos C. Machado machado@ualberta.ca
Richard S. Sutton sutton@cs.ualberta.ca
Reinforcement Learning and Artificial Intelligence Laboratory
Department of Computing Science
University of Alberta
T6G 2E8, Canada

Editor:

Abstract

The temporal-difference methods TD(λ) and Sarsa(λ) form a core part of modern reinforcement learning. Their appeal comes from their good performance, low computational cost, and their simple interpretation, given by their forward view. Recently, new versions of these methods were introduced, called true online TD(λ) and true online Sarsa(λ), respectively (van Seijen and Sutton, 2014). Algorithmically, these true online methods only make two small changes to the update rules of the regular methods, and the extra computational cost is negligible in most cases. However, they follow the ideas underlying the forward view much more closely. In particular, they maintain an exact equivalence with the forward view at all times, whereas the traditional versions only approximate it for small step-sizes. We hypothesize that these true online methods not only have better theoretical properties, but also dominate the regular methods empirically. In this article, we put this hypothesis to the test by performing an extensive empirical comparison. Specifically, we compare the performance of true online TD(λ)/Sarsa(λ) with regular TD(λ)/Sarsa(λ) on random MRPs, a real-world myoelectric prosthetic arm, and a domain from the Arcade Learning Environment. We use linear function approximation with tabular, binary, and non-binary features. Our results suggest that the true online methods indeed dominate the regular methods. Across all domains/representations the learning speed of the true online methods are often better, but never worse than that of the regular methods. An additional advantage is that no choice between traces has to be made for the true online methods. We show that new true online temporal-difference methods can be derived by making changes to the real-time forward view and then rewriting the update equations.

1. Introduction

Temporal-difference (TD) learning is a core learning technique in modern reinforcement learning (Sutton, 1988; Kaelbling et al., 1996; Sutton and Barto, 1998; Szepesvári, 2010). One of the main challenges in reinforcement learning is to make predictions, in an initially unknown environment, about the (discounted) sum of future rewards, the *return*, based on currently observed feature values and a certain behaviour policy. With TD learning it is

possible to learn good estimates of the expected return quickly by bootstrapping from other expected-return estimates. TD(λ) (Sutton, 1988) is a popular TD algorithm that combines basic TD learning with *eligibility traces* to further speed learning. The popularity of TD(λ) can be explained by its simple implementation, its low-computational complexity and its conceptually straightforward interpretation, given by its forward view. The forward view of TD(λ) is that the estimate at each time step is moved toward an update target known as as the λ-*return*, where the λ-parameter determines the trade-off between bias and variance of the update target. This trade-off has a large influence on the speed of learning and its optimal setting varies from domain to domain. The ability to improve this trade-off by adjusting the value of λ is what underlies the performance advantage of eligibility traces.

Although the forward view provides a clear intuition, TD(λ) closely approximates the forward view only for appropriately small step-sizes. Until recently, this was considered an unfortunate, but unavoidable part of the theory behind TD(λ). This changed with the introduction of true online TD(λ)(van Seijen and Sutton, 2014), which allows for full control over the bias-variance trade-off at any step-size. In particular, true online TD(1) can achieve fully unbiased updates. Moreover, true online TD(λ) only requires small modifications to the TD(λ) update equations, and the extra computational cost is negligible in most cases.

We hypothesize that true online TD(λ), and its control version true online Sarsa(λ), not only have better theoretical properties than their regular counterparts, but also dominate them empirically. We test this hypothesis by performing an extensive empirical comparison between true online TD(λ), TD(λ) with accumulating traces and TD(λ) with replacing traces, as well as true online Sarsa(λ) and Sarsa(λ) (with accumulating and replacing traces). The domains we use include random Markov reward processes, a real-world myoelectric prosthetic arm, and a domain from the Arcade Learning Environment (Bellemare et al., 2013). The representations we consider range from tabular values to linear function approximation with binary and non-binary features.

Besides the empirical study, we show how true online TD(λ) can be derived. The derivation is based on an extended version of the forward view. Whereas the updates of the traditional forward view can only be computed at the end of an episode, the updates of this extended forward view can be computed in real-time, making it applicable even to non-episodic tasks. By rewriting the updates of this real-time forward view, the true online TD(λ) updates can be derived. This derivation forms a blueprint for the derivation of other true online methods. By making variations to the real-time forward view and following the same derivation as for true online TD(λ), we derive several other true online methods.

This article is organized as follows. We start by presenting the required background on Markov decision processes and introducing TD(λ), true online TD(λ), and true online Sarsa(λ). We then present our empirical study. After this study, we analyze on what type of domains a large performance difference can be expected. This is followed by the introduction of the real-time forward view and the derivation of true online TD(λ). Finally, we present several other true online methods.

2. Markov Decision Processes

Here, we present the main learning framework. As a convention, we indicate random variables by capital letters (e.g., S_t, R_t), vectors by bold letters (e.g., $\boldsymbol{\theta}$, $\boldsymbol{\phi}$), functions by lowercase letters (e.g., v), and sets by calligraphic font (e.g., $\mathcal{S}$, $\mathcal{A}$).

Reinforcement learning (RL) problems are often formalized as *Markov decision processes* (MDPs), which can be described as 5-tuples of the form $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, consisting of $\mathcal{S}$, the set of all states; $\mathcal{A}$, the set of all actions; $p(s'|s,a)$, the transition probability function, giving for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ the probability of a transition to state $s' \in \mathcal{S}$ at the next step; $r(s,a,s')$, the reward function, giving the expected reward for a transition from (s,a) to s'. γ is the discount factor, specifying how future rewards are weighted with respect to the immediate reward. Some MDPs contain *terminal states*, which divide the sequence of state transitions into *episodes*. When a terminal state is reached the current episode ends and the state is reset to the initial state.

The return at time t is defined as the discounted sum of rewards, observed after t:

$$G_t = R_{t+1} + \gamma\, R_{t+2} + \gamma^2\, R_{t+3} + ... = \sum_{i=1}^{\infty} \gamma^{i-1}\, R_{t+i}\,,$$

where R_{t+1} is the reward received after taking action A_t in state S_t. For an episodic MDP, the return is defined as the discounted sum of rewards until the end of the episode:

$$G_t = \sum_{i=1}^{T-t} \gamma^{i-1} R_{t+i}\,,$$

where T is the time step that the terminal state is reached.

Actions are taken at discrete time steps $t = 0, 1, 2, ...$ according to a *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, which defines for each action the selection probability conditioned on the state. Each policy π has a corresponding state-value function $v_\pi(s)$, which maps each state $s \in \mathcal{S}$ to the expected value of the *return* G_t from that state, when following policy π:

$$v_\pi(s) = \mathbb{E}\{G_t \,|\, S_t = s, \pi\}\,.$$

In addition, the action-value function $q_\pi(s,a)$ gives the expected return for policy π, given that action $a \in \mathcal{A}$ is taken in state $s \in \mathcal{S}$:

$$q_\pi(s,a) = \mathbb{E}\{G_t \,|\, S_t = s, A_t, = a, \pi\}\,.$$

A core task in RL is that of estimating the state-value function v_π of some policy π from data. In general, the learner does not have access to state s directly, but can only observe a feature vector $\boldsymbol{\phi}(s) \in \mathbb{R}^n$. We estimate the value function using linear function approximation, in which case the value of a state s is the inner product between a weight vector $\boldsymbol{\theta}$ and its feature vector $\boldsymbol{\phi}(s)$:

$$\hat{v}(s, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s) = \sum_{i=1}^{n} \theta_i\, \phi_i(s)\,.$$

If s is a terminal state, then by definition $\boldsymbol{\phi}(s) := \mathbf{0}$, and hence $\hat{v}(s, \boldsymbol{\theta}) = 0$. As a shorthand, we will indicate $\boldsymbol{\phi}(S_t)$, the feature vector of the state visited at time step t, by $\boldsymbol{\phi}_t$. Similarly, the action-value function q_π can be estimated using linear function approximation. In this case, the estimate is the inner product between a weight vector and an action-feature vector $\boldsymbol{\psi}(s, a)$:

$$\hat{q}(s, a, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{\psi}(s, a) = \sum_{i=1}^{n} \theta_i \, \psi_i(s, a) \, .$$

If s is a terminal state, then by definition $\boldsymbol{\psi}(s, a) := \mathbf{0}$ for all actions a. As a convention, we will use $\boldsymbol{\psi}$ to indicate action-feature vectors and $\boldsymbol{\phi}$ to indicate state-feature vectors. As a shorthand, we will indicate $\boldsymbol{\psi}(S_t, A_t)$ by $\boldsymbol{\psi}_t$.

A general model-free update rule for linear function approximation is:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \, [U_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t] \boldsymbol{\phi}_t \, , \tag{1}$$

where U_t, the *update target*, is some estimate of the expected return at time step t. There are many ways to construct an update target. For example, the TD(0) update target is:

$$U_t = R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} \, . \tag{2}$$

Update (1) is referred to as an *online* update, meaning that the weight vector changes at every time step t. Alternatively, an update target can be used for *offline updating*. In this case, the weight vector stays constant during an episode, and instead all weight corrections are added at once at the end of the episode. Online updating not only has the advantage that it can be applied to non-episodic tasks, but it will generally produce better value-function estimates, even when only considering the estimates at the end of an episode (see Sutton & Barto, Sections 7.1–3). Hence, offline updating is primarily used as an analytical tool; it is rarely used in practise.

3. Algorithms

In this Section, we present the algorithms that we will compare: TD(λ) with accumulating as well as replacing traces, and true online TD(λ). We also present the control version of true online TD(λ): true online Sarsa(λ). Finally, we discuss several other variations of TD(λ).

3.1 Conventional TD(λ)

The conventional TD(λ) algorithm is defined by the following update equations:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t & (3) \\ \mathbf{e}_t &= \gamma \lambda \mathbf{e}_{t-1} + \boldsymbol{\phi}_t & (4) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \delta_t \, \mathbf{e}_t & (5) \end{aligned}$$

for $t \geq 0$, and with $\mathbf{e}_{-1} = \mathbf{0}$. The scalar δ_t is called the *TD error*. The vector $\mathbf{e}_t$ is called the *eligibility-trace* vector, and the parameter $\lambda \in [0, 1]$ is called the *trace-decay* parameter. The update of $\mathbf{e}_t$ shown above is referred to as the *accumulating-trace* update.

Algorithm 1 accumulate TD(λ)

INPUT: $\alpha, \lambda, \gamma, \boldsymbol{\theta}_{init}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init}$
Loop (over episodes):
    obtain initial $\boldsymbol{\phi}$
    $\boldsymbol{e} \leftarrow \mathbf{0}$
    While terminal state has not been reached, do:
        obtain next feature vector $\boldsymbol{\phi}'$ and reward R
        $\delta \leftarrow R + \gamma\, \boldsymbol{\theta}^\top \boldsymbol{\phi}' - \boldsymbol{\theta}^\top \boldsymbol{\phi}$
        $\boldsymbol{e} \leftarrow \gamma\lambda \boldsymbol{e} + \boldsymbol{\phi}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta \boldsymbol{e}$
        $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi}'$

As a shorthand, we will refer to this version of TD(λ) as 'accumulate TD(λ)'. Algorithm 1 shows the corresponding pseudocode.

Accumulate TD(λ) can be very sensitive with respect to the α and λ parameters. Especially, a large value of λ combined with a large value of α can easily cause divergence, even on simple tasks with bounded rewards. For this reason, a variant of TD(λ) is often used that is more robust with respect to these parameters. This variant, which assumes binary features, uses a different trace-update equation:

$$\boldsymbol{e}_t(i) = \begin{cases} \gamma\lambda \boldsymbol{e}_{t-1}(i) & \text{if } \boldsymbol{\phi}_t(i) = 0 \\ 1 & \text{if } \boldsymbol{\phi}_t(i) = 1 \end{cases} \qquad \text{for all features } i\,.$$

This is referred to as the *replacing-trace* update. In this article, we use a simple generalization of this update rule that allows us to apply it to domains with non-binary features as well:

$$\boldsymbol{e}_t(i) = \begin{cases} \gamma\lambda \boldsymbol{e}_{t-1}(i) & \text{if } \boldsymbol{\phi}_t(i) = 0 \\ \boldsymbol{\phi}_t(i) & \text{if } \boldsymbol{\phi}_t(i) \neq 0 \end{cases} \qquad \text{for all features } i\,. \tag{6}$$

Note that for binary features this generalized trace update reduces to the default replacing-trace update. We will refer to the version of TD(λ) that uses Equation 6 as 'replace TD(λ)'.

3.2 True Online TD(λ)

The true online TD(λ) update equations are:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t & (7) \\ \boldsymbol{e}_t &= \gamma\lambda \boldsymbol{e}_{t-1} + \boldsymbol{\phi}_t - \alpha\gamma\lambda [\boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_t]\, \boldsymbol{\phi}_t & (8) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha\delta_t\, \boldsymbol{e}_t + \alpha[\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t][\boldsymbol{e}_t - \boldsymbol{\phi}_t] & (9) \end{aligned}$$

for $t \geq 0$, and with $\boldsymbol{e}_{-1} = \mathbf{0}$. Compared to accumulate TD(λ) (equations (3), (4) and (5)), both the trace update and the weight update have an additional term. We call a trace updated in this way a *dutch trace*; we call the term $\alpha[\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t][\boldsymbol{e}_t - \boldsymbol{\phi}_t]$ the

TD-error time-step correction, or simply the δ-correction. Algorithm 2 shows pseudocode that implements equations (7), (8) and (9).[1]

Algorithm 2 true online TD(λ)

INPUT: $\alpha, \lambda, \gamma, \boldsymbol{\theta}_{init}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init},\ \hat{v}_{old} \leftarrow 0$
Loop (over episodes):
    obtain initial $\boldsymbol{\phi}$
    $\boldsymbol{e} \leftarrow \mathbf{0}$
    While terminal state has not been reached, do:
        obtain next feature vector $\boldsymbol{\phi}'$ and reward R
        $\hat{v} \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\phi}$
        $\hat{v}' \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\phi}'$
        $\delta \leftarrow R + \gamma\, \hat{v}' - \hat{v}$
        $\boldsymbol{e} \leftarrow \gamma\lambda \boldsymbol{e} + \boldsymbol{\phi} - \alpha\gamma\lambda(\boldsymbol{e}^\top \boldsymbol{\phi})\, \boldsymbol{\phi}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\delta + \hat{v} - \hat{v}_{old})\boldsymbol{e} - \alpha(\hat{v} - \hat{v}_{old})\boldsymbol{\phi}$
        $\hat{v}_{old} \leftarrow \hat{v}'$
        $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi}'$

3.3 Computational Comparison

Using the pseudocode and update equations, we can compare the computational cost of the three versions of TD(λ). Let n be the total number of features and m the number of features with a non-zero value. Then, the number of basic operations (addition and multiplication) per time step for accumulate TD(λ) is $3n + 5m$. For replace TD(λ) this number is $3n + 4m$ (the replacing trace update takes $(n - m) + m$ operations, instead of $n + m$ for an accumulating trace). True online TD(λ) takes $3n + 11m$ operations in total (computing and subtracting the vector $\alpha\gamma\lambda(\boldsymbol{e}^\top \boldsymbol{\phi})\, \boldsymbol{\phi}$ requires $4m$ operations; adding the δ-correction requires $2m$ operations). Hence, if sparse feature vectors are used (that is, if $m << n$) the computational overhead of true online TD(λ) is minimal compared to accumulate/replace TD(λ). If non-sparse feature vectors are used (that is, if $m = n$), accumulate TD(λ), replace TD(λ) and true-online TD(λ) require $8n$, $7n$ and $14n$ operations, respectively. So in this case, true online TD(λ) is roughly twice as expensive as conventional TD(λ).

3.4 True Online Sarsa(λ)

TD(λ) and true online TD(λ) are policy evaluation methods. However, they can be turned into control methods in a straightforward way. From a learning perspective, the main difference is that an estimate of the action-value function q_π should be learned, rather than of the state-value function v_π. In other words, action feature-vectors instead of state feature-vectors have to be used. Another difference is that instead of having a fixed policy

1. We provide pseudocode for true online TD(λ) with time-dependent step-size in Section 7.1. For reasons explained in that section, this requires a modified trace update. In addition, for reference purposes, we provide pseudocode for the special case of tabular features in Section 7.3.

that generates the behaviour, the policy depends on the action-value estimates. Because these estimates typically improve over time, so does the policy. The (on-policy) control counterpart of TD(λ) is the popular Sarsa(λ) algorithm. The control counterpart of true online TD(λ) is 'true online Sarsa(λ)'. Algorithm 3 shows pseudocode for true online Sarsa(λ).

Algorithm 3 true online Sarsa(λ)

INPUT: $\alpha, \lambda, \gamma, \boldsymbol{\theta}_{init}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init}, \quad \hat{q}_{old} \leftarrow 0$
Loop (over episodes):
  obtain initial state S
  select action A based on state S (for example ϵ-greedy)
  $\boldsymbol{\psi} \leftarrow$ features corresponding to S, A
  $\boldsymbol{e} \leftarrow \mathbf{0}$
  While terminal state has not been reached, do:
    take action A, observe next state S' and reward R
    select action A' based on state S'
    $\boldsymbol{\psi}' \leftarrow$ features corresponding to S', A' (if S' is terminal state, $\boldsymbol{\psi}' \leftarrow \mathbf{0}$)
    $\hat{q} \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\psi}$
    $\hat{q}' \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\psi}'$
    $\delta \leftarrow R + \gamma\, \hat{q}' - \hat{q}$
    $\boldsymbol{e} \leftarrow \gamma\lambda \boldsymbol{e} + \boldsymbol{\psi} - \alpha\gamma\lambda [\boldsymbol{e}^\top \boldsymbol{\psi}]\, \boldsymbol{\psi}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\delta + \hat{q} - \hat{q}_{old})\, \boldsymbol{e} - \alpha(\hat{q} - \hat{q}_{old}) \boldsymbol{\psi}$
    $\hat{q}_{old} \leftarrow \hat{q}'$
    $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi}'\,;\ A \leftarrow A'$

To ensure accurate estimates for all state-action values are obtained, some exploration strategy has to be used. A simple, but often sufficient strategy is to use an ϵ-greedy behaviour policy. That is, given current state S_t, with probability ϵ a random action is selected, and with probability $1 - \epsilon$ the greedy action is selected:

$$A_t^{greedy} = \operatorname*{argmax}_a \boldsymbol{\theta}_t^\top \boldsymbol{\psi}(S_t, a)\,.$$

A common way to derive an action-feature vector $\boldsymbol{\psi}(s, a)$ from a state-feature vector $\boldsymbol{\phi}(s)$ involves an action-feature vector of size $n|\mathcal{A}|$, where n is the number of state features and $|\mathcal{A}|$ is the number of actions. Each action corresponds with a block of n features in this action-feature vector. The features in $\boldsymbol{\psi}(s, a)$ that correspond to action a take on the values of the state features; the features corresponding to other actions have a value of 0.

3.5 Other Variations on TD(λ)

Several variations on TD(λ) other than those treated in this paper have been suggested in the literature. Schapire and Warmuth (1996) introduced a variation of TD(λ) for which upper and lower bounds on performance can be derived and proven. Maei, Szepesvari, Sutton, and others (Maei, 2011; Sutton et al., 2009a,b, 2014) have explored generalizations of TD(λ)-like algorithms to *off-policy learning*, in which the behavior policy (generating the

data) and the evaluation policy (whose value function is being learned) are allowed to be different.

4. Empirical Study

This section contains our main empirical study, comparing TD(λ), as well as Sarsa(λ), with their true online counterparts. For each method and each domain, a scan over the step-size α and the trace-decay parameter λ is performed such that the optimal performance can be determined. In Section 4.4, we discuss the results.

4.1 Random MRPs

For our first series of experiments we used randomly constructed Markov Reward Processes (MRPs).[2] An MRP can be interpreted as an MDP with only a single action per state (consequently, there is only one policy possible). We represent a random MRP as a 3-tuple (k, b, σ), consisting of k, the number of states; b, the branching factor (that is, the number of possible next states per transition); and σ, the standard deviation of the reward. The next states for a particular state are drawn from the total set of states at random, and without replacement. The transition probabilities to those states are randomized as well (by partitioning the unit interval at $b-1$ random cut points). The expected value of the reward for a transition is drawn from a normal distribution with zero mean and unit variance. The actual reward is drawn from a normal distribution with mean equal to this expected reward and standard deviation σ. Our random MRPs do not contain terminal states.[3]

We compared the performance of TD(λ) on three different MRPs: one with a small number of states, $(10, 3, 0.1)$, one with a large number of states, $(100, 10, 0.1)$, and one with a large number of states but a low branching factor and no stochasticity in reward generation, $(100, 3, 0)$. $\gamma = 0.99$ for all three MRPs. Each MRP is evaluated using three different representations. The first representation consists of *tabular* features, that is, each state is represented with a unique standard-basis vector of k dimensions. The second representation is based on *binary* features. The binary representation is constructed by first assigning indices, from 1 to k, to all states. Then, the binary encoding of the index of a state is used as a feature vector to represent that state. The length of a feature vector is determined by the total number of states: for $k = 10$, the length is 4; for $k = 100$, the length is 7. As an example, for $k = 10$ the feature vectors of states 1, 2 and 3 are $(0, 0, 0, 1)$,$(0, 0, 1, 0)$ and $(0, 0, 1, 1)$, respectively. Finally, the third representation uses non-binary, normalized features. For this representation each state is mapped to a 5-dimensional feature vector, with the value of each feature drawn from a normal distribution with zero mean and unit variance. After all the feature values for a state are drawn, they are normalized such that the feature vector has unit length. Once generated, the feature vectors are kept fixed for each state. We refer to this last representation as the *normal* representation.

2. The process we used to construct these MRPs is based on the process used by Bhatnagar, Sutton, Ghavamzadeh and Lee (2009).
3. The code for the MRP experiments is published online at: https://github.com/armahmood/totd-rndmdp-experiments

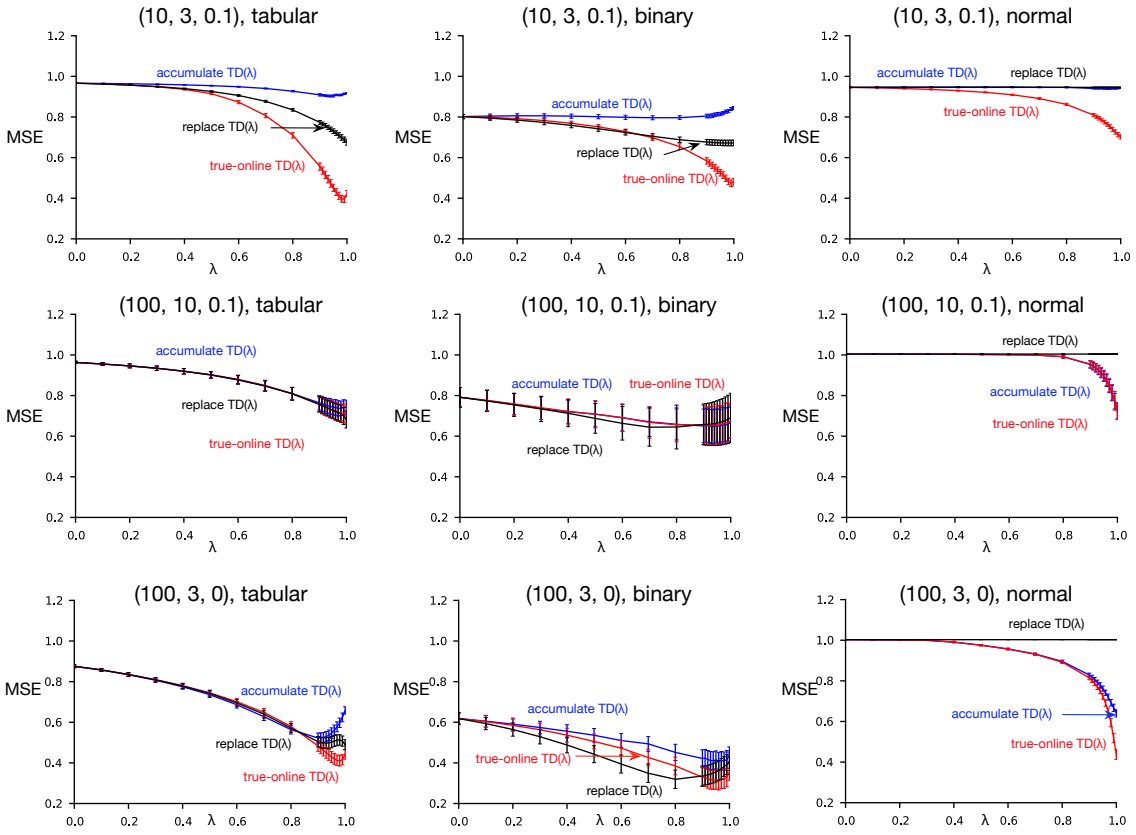

Figure 1: MSE error during early learning for three different MRPs, indicated by (k, b, σ), and three different representations. The error shown is at optimal α value.

In each experiment, we performed a scan over α and λ. Specifically, between 0 and 0.1, α is varied according to 10^i with i varying from -3 to -1 with steps of 0.2, and from 0.1 to 2.0 (linearly) with steps of 0.1. In addition, λ is varied from 0 to 0.9 with steps of 0.1 and from 0.9 to 1.0 with steps of 0.01. The initial weight vector is the zero vector in all domains. As performance metric we used the mean-squared error (MSE) with respect to the LMS solution during early learning (for k = 10, we averaged over the first 100 time steps; for k = 100, we averaged over the first 1000 time steps). We normalized this error by dividing it by the MSE under the initial weight estimate.

Figure 1 shows the results for different λ at the best value of α. In Appendix A, the results for all α values are shown. A number of observations can be made. First of all, the straightforward generalization of the replacing-trace update rule, Equation (6), is not effective. For all three domains, when replacing traces are combined with normal features, all λ values result in the same performance. The reason is that normal features practically never become zero, and hence $\boldsymbol{e}_t = \boldsymbol{\phi}_t$ almost all the time. A second observation is that the optimal performance of true online TD(λ) is, on all domains and for all representations, at

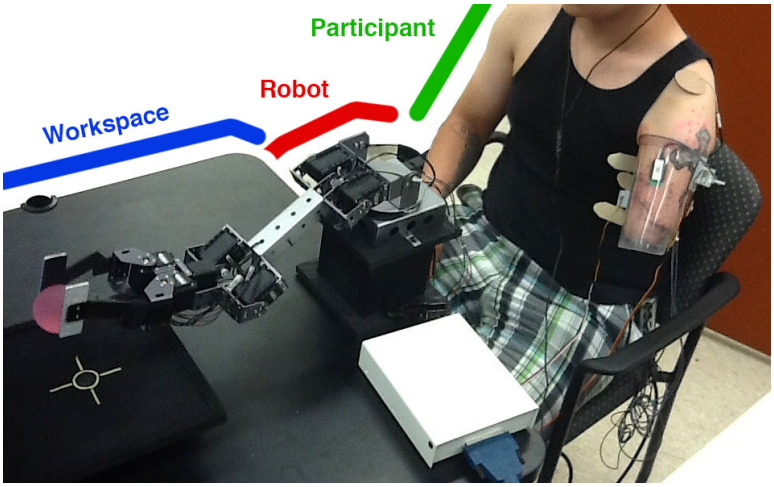

Figure 2: Source of the input data stream and predicted signals used in this experiment: a participant with an amputation performing a simple grasping task using a myoelectrically controlled robot arm, as described in Pilarski et al. (2013). More detail on the subject and experimental setting can be found in Hebert et al. (2014).

least as good as the optimal performance of accumulate TD(λ) or replace TD(λ). A more in-dept discussion of these results is provided in Section 4.4.

4.2 Predicting Signals from a Myoelectric Prosthetic Arm

In this experiment, we compared the performance of true online TD(λ) and TD(λ) on a real-world data-set consisting of sensorimotor signals measured during the human control of an electromechanical robot arm. The source of the data is a series of manipulation tasks performed by a participant with an amputation, as presented by Pilarski et al. (2013). In this study, an amputee participant used signals recorded from the muscles of their residual limb to control a robot arm with multiple degrees-of-freedom (Figure 2). Interactions of this kind are known as *myoelectric control* (c.f., Parker et al., 2006).

For consistency and comparison of results, we used the same source data and prediction learning architecture as published in Pilarski et al. (2013). In total, two signals are predicted: grip force and motor angle signals from the robot's hand. Specifically, the target for the prediction is a discounted sum of each signal over time, similar to return predictions (c.f., general value functions and nexting; Sutton et al., 2011; Modayil et al., 2014). Where possible, we used the same implementation and code base as Pilarski et al. (2013). Data for this experiment consisted of 58,000 time steps of recorded sensorimotor information, sampled at 40 Hz (i.e., approximately 25 minutes of experimental data). The state space consisted of a tile-coded representation of the robot gripper's position, velocity, recorded gripping force, and two muscle contraction signals from the human user. A standard implementation of tile-coding was used, with ten bins per signal, eight overlapping tilings, and a single active bias unit. This results in a state space with 800,001 features, 9 of which were active at any given time. Hashing was used to reduce this space down to a vector of

200,000 features that are then presented to the learning system. All signals were normalized between 0 and 1 before being provided to the function approximation routine. The discount factor for predictions of both force and angle was $\gamma = 0.97$, as in the results presented by Pilarski et al. (2013). Parameter sweeps over λ and α are conducted for all three methods. The performance metric is the mean absolute return error over all 58,000 time steps of learning, normalized by dividing by the error for $\lambda = 0$.

Figure 13 shows the performance for the angle as well as the force predictions at the best α value for different values of λ. In Appendix B, the results for all α values are shown. The relative performance of replace TD(λ) and accumulate TD(λ) depends on the predictive question being asked. For predicting the robot's grip force signal—a signal with small magnitude and rapid changes—replace TD(λ) is better than accumulate TD(λ) at all non-zero λ values. However, for predicting the robot's hand actuator position, a smoothly changing signal that varies between a range of ∼300-500, accumulate TD(λ) dominates replace TD(λ) over all non-zero λ values. True online TD dominates both methods for all non-zero λ values on both prediction tasks (force and angle).

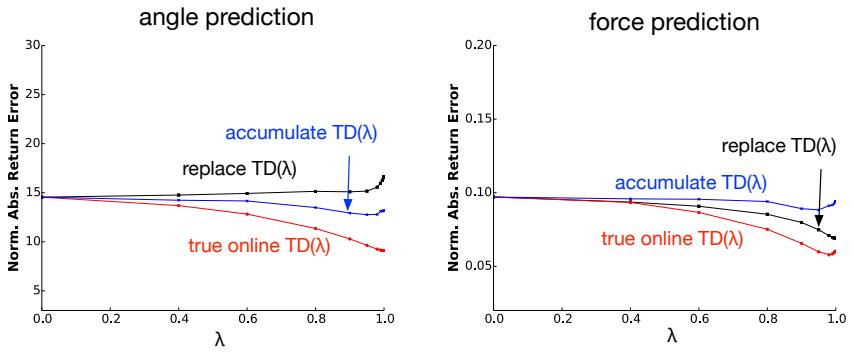

Figure 3: Performance as function of λ at the optimal α value, for the prediction of the servo motor angle (left), as well as the grip force (right).

4.3 Control in the ALE Domain Asterix

In this experiment, we compared the performance of true online Sarsa(λ) with that of accumulate Sarsa(λ) and replace Sarsa(λ), on a domain from the Arcade Learning Environment (ALE) (Bellemare et al., 2013; Defazio and Graepel, 2014; Mnih et al., 2015), called Asterix.[4] The ALE is a general testbed that provides an interface to hundreds of Atari 2600 games in which one has access, at each frame, to the game screen, the current RAM state and to a reward signal obtained from the transition between game frames. At each frame the agent provides one of the 18 possible actions in the game (equivalent to the 18 different actions allowed in the joystick) with the goal of maximizing the (discounted) sum of rewards.

4. We used ALE version 0.4.4 for our experiments. The code for the ALE experiments is published online at: https://github.com/mcmachado/TrueOnlineSarsa

In the Asterix domain (see Figure 4 for a screenshot), the agent controls a yellow avatar, which has to collect 'potion' objects, while avoiding 'harp' objects. Both potions and harps move across the screen horizontally. Every time the agent collects a potion it receives a reward of 50 points, and every time it touches a harp it looses a life (it has three lives in total). The game ends after the agent has lost three lives, or after 5 minutes, whichever comes first.[5]

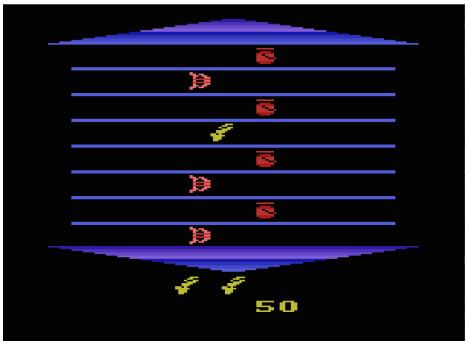

Figure 4: Screenshot of the game Asterix.

The agent can use the actions *up, right, down,* and *left* to move across the screen, a *no-op* action, as well as combinations of two directions, resulting in a diagonal move (e.g.*up-right*). This results in 9 actions in total. The state-space representation is based on linear function approximation. We use what Bellemare et al. (2013) called *Basic* feature set, which "encodes the presence of colours on the Atari 2600 screen." It is obtained by first subtracting the game screen background (see Bellemare et al., 2013, sec. 3.1.1) and then dividing the remaining screen in to 16×14 tiles of size 10×15 pixels. Finally, for each tile, one binary feature is generated for each of the 128 available colours, encoding whether a colour is active or not in that tile. This generates 28,672 features (besides a bias term that is always active).

Because episode lengths can vary hugely (basically, from about 10 seconds all the way up to 5 minutes), constructing a fair performance metric is non-trivial. For example, comparing the average return on the first N episodes of two methods is only fair if they have seen roughly the same amount of samples in those episodes, which is not guaranteed for this domain. On the other hand, looking at the total reward collected for the first X samples is also not a good metric, because there is no negative reward associated to dying. To resolve this, we look at the return per episode, averaged over the first n(X) episodes, where n(X) is the number of episodes observed in the first X samples. More specifically, our metric consists of the average score per episode while learning for 20 hours (4,320,000 frames). In addition, we averaged the resulting number over 400 independent runs.

As with the evaluation experiments, we performed a scan over the step-size α and the trace-decay parameter λ. Specifically, we looked at all combinations of $\alpha \in \{0.20, 0.50, 0.80, 1.10, 1.40, 1.70, 2.00\}$ and $\lambda \in \{0.00, 0.50, 0.80, 0.90, 0.95, 0.99\}$ (these values were determined during a preliminary parameter sweep). We used a discount factor $\gamma = 0.999$ and

5. We added the 5 minute time limit ourselves as in previous work (Bellemare et al., 2013); the original game has no time limit.

ϵ-greedy exploration with $\epsilon = 0.01$. The weight vector was initialized to the zero vector. Also, as Bellemare et al. (2013) , we take an action at each 5 frames, this decreases the algorithms running time and it also tries to avoid "super-human" reflexes in our agents.

The results are shown in Figure 5. On this domain, the optimal performance of all three versions of Sarsa(λ) is similar.

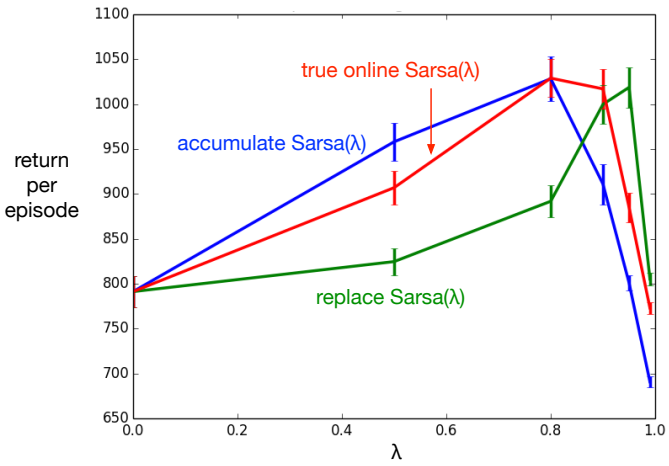

Figure 5: Return per episode, averaged over the first 4,320,000 frames as well as 400 independent runs, as function of λ, at optimal α, on the Asterix domain.

4.4 Discussion

Figure 6 summarizes the performance of the different TD(λ) versions on all evaluation domains. Specifically, it shows the error for each method at its best settings of α and λ. The error is normalized by dividing it by the error at $\lambda = 0$ (remember that all versions of TD(λ) behave the same for $\lambda = 0$). Because $\lambda = 0$ lies in the parameter range that is being optimized over, the normalized error can never be higher than 1. If for a method/domain the normalized error is equal to 1, this means that setting λ higher than 0 either has no effect, or that the error gets worse. In either case, eligibility traces are not effective for that method/domain.

Overall, true online TD(λ) is clearly better than accumulate TD(λ) and replace TD(λ) in terms of optimal performance. Specifically, on each considered domain, the error for true online TD(λ) is either smaller or equal to the error of accumulate/replace TD(λ). This is especially impressive, given the wide of variety of domains, and the fact the computational overhead for true online TD(λ) is small (see Section 3.3 for details).

Comparing accumulate TD(λ) with replace TD(λ), it can be seen that, when considering tabular or binary features, on some domains accumulate TD(λ) performs best, while on others replace TD(λ) performs best. When normal features are used, our naive generalization of replace TD(λ) is not effective (standard replace TD(λ) is not defined for normal features).

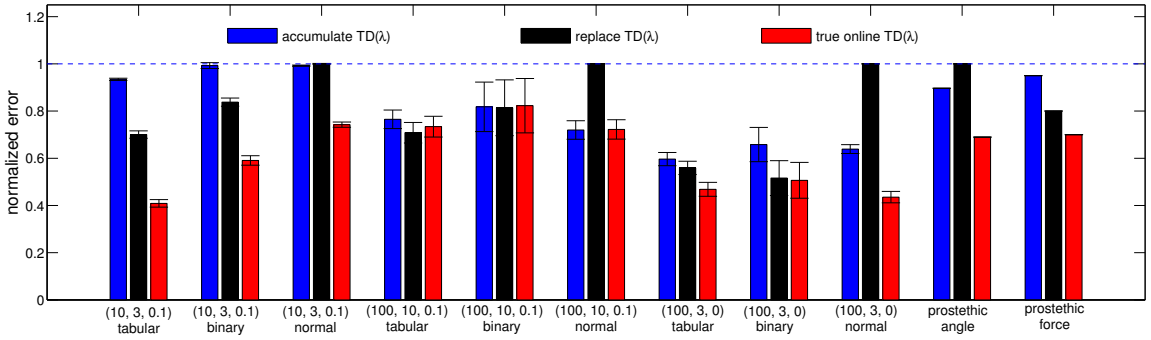

Figure 6: Summary of the evaluation results: error at optimal (α, λ)-settings for all domains, normalized with the TD(0) error.

On the Asterix domain, the performance of the three Sarsa(λ) versions is similar. This is in accordance with the evaluation results, which showed that the size of the performance difference is domain dependent. In the worst case, the performance of the true online method is similar to that of the regular method.

The optimal performance is not the only factor that determines how good a method is; what also matters is how easy it is to find this performance. The detailed plots in Appendix A and B reveal that the parameter sensitive of accumulate TD(λ) is much higher than that of true online TD(λ) and replace TD(λ). This is clearly visible in the first MRP task (Figure 10), as well as the experiments with the myoelectric prosthetic arm (Figure 13).

There is one more thing to take away from the experiments. In the first MRP, (10, 3, 0.1), with normal features, accumulate TD(λ), as well as replace TD(λ), are ineffective (see Figure 6: the normalized performance of accumulate/replace TD(λ) is 1, meaning that the performance at optimized λ is equal to the performance of TD(0)). However, true online TD(λ) was able to obtain a considerable performance advantage with respect to TD(0). This demonstrates that true online TD(λ) expands the set of domains/representations where eligibility traces are effective. This could potentially have far-reaching consequences. Specifically, using non-binary features becomes a lot more interesting. Replacing traces are not feasible / ineffective for such representations, while using accumulating traces can easily result in divergence of values. However, for true online TD(λ) non-binary features are not necessarily more challenging than binary features. Exploring new, non-binary representations could potentially further improve the performance for true online TD(λ) on domains such as the myoelectic prosthetic arm or the Asterix domain.

5. Analytical Comparison

The empirical study suggests that true online TD(λ) performs at least as good as accumulate TD(λ) and replace TD(λ). In this section, we try to answer the question on what kind of domains a large difference in performance can be expected, and similarly, when no difference is expected. The following three theorems provide some insights into this.

Theorem 1 *For $\lambda = 0$, accumulate TD(λ), replace TD(λ) and true online TD(λ) behave the same.*

Proof For $\lambda = 0$, the accumulating-trace update, the (generalized) replacing-trace update and the dutch-trace update all reduce to $\boldsymbol{e}_t = \boldsymbol{\phi}_t$. In addition, because $\boldsymbol{e}_t = \boldsymbol{\phi}_t$, the δ-correction of true online TD(λ) is 0. ■

A feature i is *visited* at time t if $\boldsymbol{\phi}_t(i) > 0$. The following theorem shows that any difference in behaviour between the three versions of TD(λ) is due to how revisits of features are handled.

Theorem 2 *When no features are revisited within the same episode, accumulate TD(λ), replace TD(λ) and true online TD(λ) behave the same (for any λ).*

Proof Because at the start of an episode all trace values are 0, and because a feature is only visited once within an episode, if $\boldsymbol{\phi}_t(i) \neq 0$, then $\boldsymbol{e}_{t-1}(i) = 0$ and if $\boldsymbol{e}_{t-1}(i) \neq 0$, then $\boldsymbol{\phi}_t(i) = 0$. Hence, the accumulating-trace update and the generalized replacing-trace update have the same effect. It also means that $\boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_t$ is always zero. Hence, the dutch-trace update reduces to the accumulating-trace update. In addition, because the weight of a feature does not get updated until the feature is visited, if $\boldsymbol{\phi}_t(i) \neq 0$, then $\boldsymbol{\theta}_t(i) - \boldsymbol{\theta}_{t-1}(i) = 0$, and if $\boldsymbol{\theta}_t(i) - \boldsymbol{\theta}_{t-1}(i) \neq 0$, then $\boldsymbol{\phi}_t(i) \neq 0$. It follows that $\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t$ is always 0, and hence the δ-correction as well. ■

Finally, our third theorem states that for small step-sizes the behaviour of true online TD(λ) approximates that of accumulate TD(λ):

Theorem 3 *Let Δ_t^{acc} be the weight update at time t due to accumulate TD(λ) and Δ_t^{true} the weight update due to true online TD(λ). If $\gamma\lambda < 1$ and the feature vectors and TD errors are bounded, then $\Delta_t^{acc}/\Delta_t^{true} \to 1$ if $\alpha \to 0$.*

Proof The update equations specify that

$$\begin{aligned} \Delta_t^{acc} &:= \alpha \boldsymbol{e}_t^{acc} \delta_t \,, \\ \Delta_t^{true} &:= \alpha \boldsymbol{e}_t^{dut} \delta_t + \alpha [\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t][\boldsymbol{e}_t^{dut} - \boldsymbol{\phi}_t] \,, \end{aligned}$$

where $\boldsymbol{e}_t^{acc}$ is an accumulating trace, and $\boldsymbol{e}_t^{dut}$ is a dutch trace. We will prove the theorem by showing that Δ_t^{true} can be written as:

$$\Delta_t^{true} = \alpha \big[\boldsymbol{e}_t^{acc} \delta_t + \boldsymbol{c}(\alpha) \big]$$

with $\boldsymbol{c}(\alpha) \to 0$ if $\alpha \to 0$. More specifically, Δ_t^{true} can be written as:

$$\Delta_t^{true} = \alpha \Big[\boldsymbol{e}_t^{acc} \delta_t + (\boldsymbol{e}_t^{dut} - \boldsymbol{e}_t^{acc}) \delta_t + (\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})^\top \boldsymbol{\phi}_t (\boldsymbol{e}_t^{dut} - \boldsymbol{\phi}_t) \Big]$$

We will show that $\boldsymbol{e}_t^{dut} - \boldsymbol{e}_t^{acc} \to 0$ if $\alpha \to 0$, and that $(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})^\top \boldsymbol{\phi}_t (\boldsymbol{e}_t^{dut} - \boldsymbol{\phi}_t) \to 0$ if $\alpha \to 0$.

The non-incremental expression for $\boldsymbol{e}_t^{acc}$ is:

$$\begin{aligned}
\boldsymbol{e}_0^{acc} &= \boldsymbol{\phi}_0 \\
\boldsymbol{e}_1^{acc} &= \gamma\lambda\boldsymbol{\phi}_0 + \boldsymbol{\phi}_1 \\
\boldsymbol{e}_2^{acc} &= (\gamma\lambda)^2\boldsymbol{\phi}_0 + \gamma\lambda\boldsymbol{\phi}_1 + \boldsymbol{\phi}_2 \\
&\vdots \\
\boldsymbol{e}_t^{acc} &= \sum_{i=0}^{t}(\gamma\lambda)^{t-i}\boldsymbol{\phi}_t
\end{aligned}$$

Let the value of feature i be bounded by C, that is $|\boldsymbol{\phi}_t(i)| < C$ for all i, t. Then, $|\boldsymbol{e}_t^{acc}(i)| < C/(1-\gamma\lambda)$ for all i, t. Because $\gamma\lambda < 1$, this is some finite value.

The dutch-trace update can be re-written as:

$$\boldsymbol{e}_t^{dut} = \gamma\lambda(\mathbf{I} - \alpha\boldsymbol{\phi}_t\boldsymbol{\phi}_t^\top)\,\boldsymbol{e}_{t-1}^{dut} + \boldsymbol{\phi}_t$$

Using this, the non-incremental expression for $\boldsymbol{e}_t^{dut}$ becomes:

$$\begin{aligned}
\boldsymbol{e}_0^{dut} &= \boldsymbol{\phi}_0 \\
\boldsymbol{e}_1^{dut} &= \gamma\lambda(\mathbf{I} - \alpha\boldsymbol{\phi}_1\boldsymbol{\phi}_1^\top)\boldsymbol{\phi}_0 + \boldsymbol{\phi}_1 \\
\boldsymbol{e}_2^{dut} &= (\gamma\lambda)^2(\mathbf{I} - \alpha\boldsymbol{\phi}_2\boldsymbol{\phi}_2^\top)(\mathbf{I} - \alpha\boldsymbol{\phi}_1\boldsymbol{\phi}_1^\top)\boldsymbol{\phi}_0 + \gamma\lambda(\mathbf{I} - \alpha\boldsymbol{\phi}_2\boldsymbol{\phi}_2^\top)\boldsymbol{\phi}_1 + \boldsymbol{\phi}_2 \\
&\vdots
\end{aligned}$$

Because the feature vectors are bounded, if $\alpha \to 0$, $(\mathbf{I} - \alpha\boldsymbol{\phi}_i\boldsymbol{\phi}_i^\top) \to \mathbf{I}$, and $\boldsymbol{e}_t^{dut} \to \boldsymbol{e}_t^{acc}$ (because the trace values are bounded, this is true even if $t \to \infty$).

Finally, we need to show that $(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})^\top\boldsymbol{\phi}_t(\boldsymbol{e}_t^{true} - \boldsymbol{\phi}_t) \to 0$ if $\alpha \to 0$. Because the feature vectors and trace values are bounded, it suffices to show that $\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1} = \Delta_{t-1}^{true} \to 0$ if $\alpha \to 0$, which follows from the definition of Δ_t^{true} (given the condition that the TD error is bounded).

■

Based on these three theorems, we expect a large difference on domains for which the optimal α and optimal λ are relatively large, and where features are frequently revisited. Domains with a relatively large optimal α and optimal λ are typically domains with relatively low stochasticity. So as a rule of thumb, a large difference can be expected on domains with relatively low stochasticity and frequent revisits of features.

6. Derivation of True Online TD(λ)

The defining property of a true online method is that it maintains an exact equivalence with an online forward view at all times. This means that at every moment in time, the weight vector can be interpreted as the result of a sequence of updates with multi-step update targets. To achieve this step-by-step equivalence, the regular forward has to be extended, because it only specifies what the weights at the end of an episode should be. In this section, we present the extended forward view, and we derive the true online TD(λ) update equations from it.

6.1 The forward view of TD(λ)

In Section 2, the general update rule for linear function approximation was presented (Equation 1), which is based on the update rule for stochastic gradient descent. The update equations for TD(λ), however, are of a different form (Equations 3, 4 and 5). The forward view of TD(λ) relates the TD(λ) equations to Equation 1. Specifically, the forward view of TD(λ) specifies that TD(λ) approximates the λ-return algorithm. This algorithm performs a series of updates of the form of Equation 1 with the λ-return as update target:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left[G_t^\lambda - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right] \boldsymbol{\phi}_t \,, \qquad \text{for } 0 \leq t < T,$$

where T is the end of the episode, and G_t^λ is the λ-return at time t.

The λ-return is a multi-step update target based on a weighted average of all future state values, with λ determining the weight distribution. Specifically, the λ-return at time t is defined as:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}(\boldsymbol{\theta}_t)$$

with $G_t^{(n)}(\boldsymbol{\theta})$ is the n-step return, defined as:

$$G_t^{(n)}(\boldsymbol{\theta}) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \, \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+n} \,.$$

For episodic tasks, $G_t^{(n)}(\boldsymbol{\theta})$ is equal to the full return, G_t, if $t + n \geq T$, and the λ-return can be written as:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)}(\boldsymbol{\theta}_t) + \lambda^{T-t-1} G_t. \tag{10}$$

The forward view offers a particularly straightforward interpretation of the λ-parameter. For $\lambda = 0$, G_t^λ reduces to the TD(0) update target, while for $\lambda = 1$, G_t^λ reduces to the full return. In other words, for $\lambda = 0$ the update target has maximum bias and minimum variance, while for $\lambda = 1$, the update target is unbiased, but has maximum variance. For λ in between 0 and 1, the bias and variance are between these two extremes. So, λ enables control over the trade-off between bias and variance.

While the λ-return algorithm has a very clear intuition, there is only an exact equivalence for the offline case. That is, the offline variant of TD(λ) computes the same value estimates as the offline variant of the λ-return algorithm. For the online case, there is only an approximate equivalence. Specifically, the weight vector at time T, computed by accumulate TD(λ) closely approximates the weight vector at time T computed by the online λ-return algorithm for appropriately small values of the step-size parameter (Sutton and Barto, 1998).

That the forward view only applies to the weight vector at the end of an episode, even in the online case, is a limitation that is often overlooked. It is related to the fact that the λ-return for S_t is constructed from data stretching from time $t+1$ all the way to time T, the time that the terminal state is reached. A consequence is that the λ-return algorithm can compute its weight vectors only in hindsight, at the end of an episode. This is illustrated by Figure 7, which maps each weight vector $\boldsymbol{\theta}_t$ to the earliest time that it can be computed. 'Time' in this case refers to the time of data-collection: time t is defined as the moment

that sample $\boldsymbol{\phi}_t$ is observed. By contrast, TD(λ) uses only data up to time t to compute the weight vector $\boldsymbol{\theta}_t$. Hence, TD(λ) can compute its weight vectors without delay (see Figure 8). To denote this important property, we use the term *real-time*. TD(λ) is a real-time algorithm, while the λ-return algorithm is not. A consequence is that even though both algorithms compute a sequence of T weight vectors, a meaningful comparison can only be made for $\boldsymbol{\theta}_T$, because only at time T does TD(λ) have access to the same data as the λ-return algorithm. This limits the usefulness of the λ-return algorithm as an intuitive way to view TD(λ). In the next section, we address this limitation.

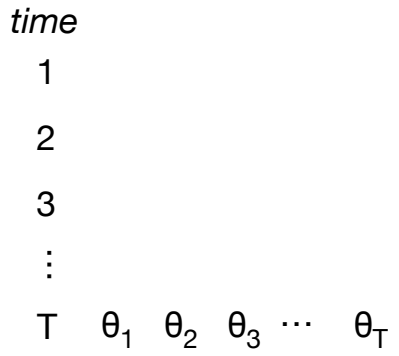

Figure 7: The weight vectors of the λ-return algorithm mapped to the earliest time that they can be computed.

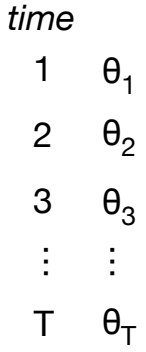

Figure 8: The weight vectors of TD(λ) mapped to the earliest time that they can be computed.

6.2 The Real-Time Forward View

The conventional forward view explains how the weight vector at the end of an episode, computed by TD(λ), can be interpreted as the result of a sequence of updates with a particular multi-step update target, the λ-return. We want to give a similar explanation for

weight vectors *during* an episode. In other words, we want to construct a real-time forward view that explains the weight vectors, computed by TD(λ), at all time steps.

The dilemma that arises when trying to construct a real-time forward view is that the update targets should contain data from many time steps ahead, but the real-time aspect prohibits the use of data beyond the current time step. The solution to this dilemma is to have update targets that grow over time. In other words, rather than defining a fixed update target for each visited state, the update target depends on the time step up to which data is observed. We call such an update target an *interim update target*, and the time step up to which data is observed the *data-horizon*. We will use a superscript to indicate the data-horizon h of an update target: U_t^h. A simple example of an interim update target is an update target that consists of the discounted sum of rewards up to the data-horizon:

$$U_t^h \quad = \quad R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{h-t-1} R_h \,.$$

A direct consequence of having update targets that depend on the data-horizon is that a real-time forward view specifies an update sequence for *each* data-horizon. Below, we show the update sequences based on an interim update target U_t^h for horizons 1, 2 and 3 ($\boldsymbol{\theta}_0^h := \boldsymbol{\theta}_{init}$, for all h).

$$\begin{aligned}
h = 1: \quad & \boldsymbol{\theta}_1^1 = \boldsymbol{\theta}_0^1 + \alpha \Big[U_0^1 - (\boldsymbol{\theta}_0^1)^\top \boldsymbol{\phi}_0 \Big] \boldsymbol{\phi}_0 \,, \\
h = 2: \quad & \boldsymbol{\theta}_1^2 = \boldsymbol{\theta}_0^2 + \alpha \Big[U_0^2 - (\boldsymbol{\theta}_0^2)^\top \boldsymbol{\phi}_0 \Big] \boldsymbol{\phi}_0 \,, \\
& \boldsymbol{\theta}_2^2 = \boldsymbol{\theta}_1^2 + \alpha \Big[U_1^2 - (\boldsymbol{\theta}_1^2)^\top \boldsymbol{\phi}_1 \Big] \boldsymbol{\phi}_1 \,, \\
h = 3: \quad & \boldsymbol{\theta}_1^3 = \boldsymbol{\theta}_0^3 + \alpha \Big[U_0^3 - (\boldsymbol{\theta}_0^3)^\top \boldsymbol{\phi}_0 \Big] \boldsymbol{\phi}_0 \,, \\
& \boldsymbol{\theta}_2^3 = \boldsymbol{\theta}_1^3 + \alpha \Big[U_1^3 - (\boldsymbol{\theta}_1^3)^\top \boldsymbol{\phi}_1 \Big] \boldsymbol{\phi}_1 \,, \\
& \boldsymbol{\theta}_3^3 = \boldsymbol{\theta}_2^3 + \alpha \Big[U_2^3 - (\boldsymbol{\theta}_2^3)^\top \boldsymbol{\phi}_2 \Big] \boldsymbol{\phi}_2 \,,
\end{aligned}$$

More generally, the update sequence for horizon h is defined by:

$$\boldsymbol{\theta}_{t+1}^h = \boldsymbol{\theta}_t^h + \alpha \Big[U_t^h - (\boldsymbol{\theta}_t^h)^\top \boldsymbol{\phi}_t \Big] \boldsymbol{\phi}_t \,, \qquad \text{for } 0 \leq t < h \,. \tag{11}$$

Figure 9 maps each weight vector to the earliest time it can be computed. Ultimately, the weight-vector sequence of interest is not the sequence at a particular horizon. Rather, it is the sequence consisting of the final weight vector at each horizon: $\boldsymbol{\theta}_1^1, \boldsymbol{\theta}_2^2, \boldsymbol{\theta}_3^2, \ldots, \boldsymbol{\theta}_T^T$. Because $\boldsymbol{\theta}_t^t$ can be computed at time t, we call the forward view a real-time forward view.

In principle, Equation (11) can be combined with any interim update target definition to form a real-time forward view. However, to get the real-time forward view that belongs to TD(λ) a horizon-dependent version of the λ-return is needed. A version of the λ-return that corresponds with horizon h should not use data beyond this horizon. In other words, the highest n-step return that should be involved is the $(h-t)$-step return. This can be achieved by replacing each n-step return with $n > h - t$ with the $(h-t)$-step return. We

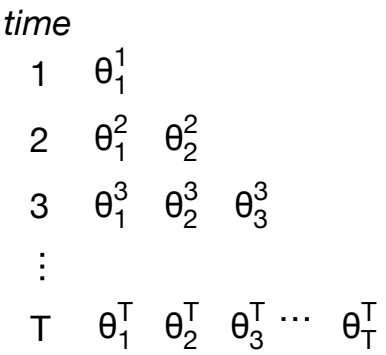

Figure 9: The weight vectors of the new forward view mapped to the earliest time that they can be computed.

call this version of the λ-return the *interim λ-return*, and use the notation $G_t^{\lambda|h}$ to indicate the interim λ-return depending on horizon h. $G_t^{\lambda|h}$ can be written as follows:

$$\begin{aligned} G_t^{\lambda|h} &= (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + (1-\lambda)\sum_{n=h-t}^{\infty}\lambda^{n-1}G_t^{(h-t)} \\ &= (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + G_t^{(h-t)}\cdot\Big[(1-\lambda)\sum_{n=h-t}^{\infty}\lambda^{n-1}\Big] \\ &= (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + G_t^{(h-t)}\cdot\Big[\lambda^{h-t-1}(1-\lambda)\sum_{k=0}^{\infty}\lambda^{k}\Big] \\ &= (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{h-t-1}G_t^{(h-t)} \end{aligned} \tag{12}$$

Equation 12 fully specifies the interim λ-return, except for one small detail: the weight vector that should be used for the value estimates in the n-step returns has not been specified yet. The regular λ-return uses $G_t^{(n)}(\boldsymbol{\theta}_t)$ (see Equation 10). For the real-time forward view, however, all weight vectors have two indices, so simply using $\boldsymbol{\theta}_t$ does not work in this case. So which double-indexed weight vector should be used? The two guiding principles on deciding which weight vector to use is that we want the forward view to be an approximation of accumulate TD(λ) and that an efficient implementation should be possible. One option is to use $G_t^{(n)}(\boldsymbol{\theta}_t^h)$. While with this definition the update-sequence at data-horizon T is exactly the same as the sequence of updates from the λ-return algorithm (basically, the λ-return implicitly uses a data-horizon of T), it prohibits efficiently computation of $\boldsymbol{\theta}_{h+1}^{h+1}$ from $\boldsymbol{\theta}_h^h$. For this reason, we use $G_t^{(n)}(\boldsymbol{\theta}_{t+n-1}^{t+n-1})$, which does allow for efficient computation, and forms a good approximation of accumulate TD(λ) as well (as we show below). Using

this weight vector, the full definition of $G_t^{\lambda|h}$ becomes:

$$G_t^{\lambda|h} := (1-\lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_t^{(n)}\left(\boldsymbol{\theta}_{t+n-1}^{t+n-1}\right) + \lambda^{h-t-1} G_t^{(h-t)}\left(\boldsymbol{\theta}_{h-1}^{h-1}\right). \tag{13}$$

We call this the *interim λ-return*. We call the algorithm that combines the interim λ-return with Equation 11 the interim λ-return algorithm.

6.3 Derivation

In this subsection, we derive the update equations of true online TD(λ) directly from the real-time forward view, defined by equations (11) and (13) (and $\boldsymbol{\theta}_0^h := \boldsymbol{\theta}_{init}$). The derivation is based on expressing $\boldsymbol{\theta}_{h+1}^{h+1}$ in terms of $\boldsymbol{\theta}_h^h$.

We start by writing $\boldsymbol{\theta}_h^h$ directly in terms of the initial weight vector and the interim λ-returns. First, we rewrite (11), with the interim λ-return as update target, as:

$$\boldsymbol{\theta}_{t+1}^h = (\mathbf{I} - \alpha \boldsymbol{\phi}_t \boldsymbol{\phi}_t^\top)\, \boldsymbol{\theta}_t^h + \alpha\, G_t^{\lambda|h}$$

with $\mathbf{I}$ the identity matrix. Now, consider $\boldsymbol{\theta}_t^h$ for $t = 1$ and $t = 2$:

$$\begin{aligned}
\boldsymbol{\theta}_1^h &= (\mathbf{I} - \alpha\boldsymbol{\phi}_0\boldsymbol{\phi}_0^\top)\boldsymbol{\theta}_{init} + \alpha\boldsymbol{\phi}_0 G_0^{\lambda|h} \\
\boldsymbol{\theta}_2^h &= (\mathbf{I} - \alpha\boldsymbol{\phi}_1\boldsymbol{\phi}_1^\top)\boldsymbol{\theta}_1^h + \alpha\boldsymbol{\phi}_1 G_1^{\lambda|h} \\
&= (\mathbf{I} - \alpha\boldsymbol{\phi}_1\boldsymbol{\phi}_1^\top)(\mathbf{I} - \alpha\boldsymbol{\phi}_0\boldsymbol{\phi}_0^\top)\boldsymbol{\theta}_{init} + \alpha(\mathbf{I} - \alpha\boldsymbol{\phi}_1\boldsymbol{\phi}_1^\top)\boldsymbol{\phi}_0 G_0^{\lambda|h} + \alpha\boldsymbol{\phi}_1 G_1^{\lambda|h}
\end{aligned}$$

For general $t \leq h$, we can write:

$$\boldsymbol{\theta}_t^h = \mathbf{A}_0^{t-1}\, \boldsymbol{\theta}_{init} + \alpha \sum_{i=1}^{t} \mathbf{A}_i^{t-1}\, \boldsymbol{\phi}_{i-1} G_{i-1}^{\lambda|h}\,,$$

where $\mathbf{A}_i^j$ is defined as:

$$\mathbf{A}_i^j := (\mathbf{I} - \alpha\boldsymbol{\phi}_j\boldsymbol{\phi}_j^\top)(\mathbf{I} - \alpha\boldsymbol{\phi}_{j-1}\boldsymbol{\phi}_{j-1}^\top)\ldots(\mathbf{I} - \alpha\boldsymbol{\phi}_i\boldsymbol{\phi}_i^\top), \quad \text{for } j \geq i\,,$$

and $\mathbf{A}_{j+1}^j := \mathbf{I}$. We are now able to express $\boldsymbol{\theta}_h^h$ as:

$$\boldsymbol{\theta}_h^h = \mathbf{A}_0^{h-1}\, \boldsymbol{\theta}_{init} + \alpha \sum_{i=1}^{h} \mathbf{A}_i^{h-1}\, \boldsymbol{\phi}_{i-1} G_{i-1}^{\lambda|h}\,, \tag{14}$$

Because for the derivation of true online TD(λ), we only need (14) and the definition of $G_t^{\lambda|h}$, we can drop the double indices for the weight vectors and use $\boldsymbol{\theta}_h := \boldsymbol{\theta}_h^h$.

We now derive a compact expression for the difference $G_t^{\lambda|h+1} - G_t^{\lambda|h}$.

$$\begin{aligned}
G_t^{\lambda|h+1} - G_t^{\lambda|h} &= (1-\lambda)\sum_{n=1}^{h-t}\lambda^{n-1}G_t^{t+n}(\boldsymbol{\theta}_{t+n-1}) + \lambda^{h-t}G_t^{h+1}(\boldsymbol{\theta}_h) \\
&\quad - (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_t^{t+n}(\boldsymbol{\theta}_{t+n-1}) - \lambda^{h-t-1}G_t^{h}(\boldsymbol{\theta}_{h-1}) \\
&= (1-\lambda)\lambda^{h-t-1}G_t^{h}(\boldsymbol{\theta}_{h-1}) + \lambda^{h-t}G_t^{h+1}(\boldsymbol{\theta}_h) - \lambda^{h-t-1}G_t^{h}(\boldsymbol{\theta}_{h-1}) \\
&= \lambda^{h-t}G_t^{h+1}(\boldsymbol{\theta}_h) - \lambda^{h-t}G_t^{h}(\boldsymbol{\theta}_{h-1}) \\
&= \lambda^{h-t}\Big[G_t^{h+1}(\boldsymbol{\theta}_h) - G_t^{h}(\boldsymbol{\theta}_{h-1})\Big] \\
&= \lambda^{h-t}\Big[\sum_{i=1}^{h+1-t}\gamma^{i-1}R_{t+i} + \gamma^{h+1-t}\boldsymbol{\theta}_h^\top\boldsymbol{\phi}_{h+1} - \sum_{i=1}^{h-t}\gamma^{i-1}R_{t+i} - \gamma^{h-t}\boldsymbol{\theta}_{h-1}^\top\boldsymbol{\phi}_h\Big] \\
&= \lambda^{h-t}\Big[\gamma^{h-t}R_{h+1} + \gamma^{h+1-t}\boldsymbol{\theta}_h^\top\boldsymbol{\phi}_{h+1} - \gamma^{h-t}\boldsymbol{\theta}_{h-1}^\top\boldsymbol{\phi}_h\Big] \\
&= (\lambda\gamma)^{h-t}\Big[R_{h+1} + \gamma\,\boldsymbol{\theta}_h^\top\boldsymbol{\phi}_{h+1} - \boldsymbol{\theta}_{h-1}^\top\boldsymbol{\phi}_h\Big]
\end{aligned}$$

Note that the difference $G_t^{\lambda|h+1} - G_t^{\lambda|h}$ is naturally expressed using a term that looks like a TD error but with a modified time step. We call this the modified TD error, δ'_h:

$$\delta'_h := R_{h+1} + \gamma\,\boldsymbol{\theta}_h^\top\boldsymbol{\phi}_{h+1} - \boldsymbol{\theta}_{h-1}^\top\boldsymbol{\phi}_h.$$

Using this definition, the difference $G_t^{\lambda|h+1} - G_t^{\lambda|h}$ can be compactly written as:

$$G_t^{\lambda|h+1} - G_t^{\lambda|h} = (\lambda\gamma)^{h-t}\delta'_h \tag{15}$$

Note that δ'_h relates to the regular TD error, δ_h, as follows:

$$\begin{aligned}
\delta'_h &= R_{h+1} + \gamma\,\boldsymbol{\theta}_h^\top\boldsymbol{\phi}_{h+1} - \boldsymbol{\theta}_{h-1}^\top\boldsymbol{\phi}_h \\
&= R_{h+1} + \gamma\,\boldsymbol{\theta}_h^\top\boldsymbol{\phi}_{h+1} - \boldsymbol{\theta}_h^\top\boldsymbol{\phi}_h + \boldsymbol{\theta}_h^\top\boldsymbol{\phi}_h - \boldsymbol{\theta}_{h-1}^\top\boldsymbol{\phi}_h \\
&= \delta_h + \boldsymbol{\theta}_h^\top\boldsymbol{\phi}_h - \boldsymbol{\theta}_{h-1}^\top\boldsymbol{\phi}_h\,.
\end{aligned} \tag{16}$$

To get the update rule, we have to express $\boldsymbol{\theta}_{h+1}$ in terms of $\boldsymbol{\theta}_h$. This is done below, using (14), (15) and (16).

$$
\begin{aligned}
\boldsymbol{\theta}_{h+1} &= \mathbf{A}_0^h\,\boldsymbol{\theta}_0 + \alpha \sum_{i=1}^{h+1} \mathbf{A}_i^h\,\boldsymbol{\phi}_{i-1} G_{i-1}^{\lambda|h+1} \\
&= \mathbf{A}_0^h \boldsymbol{\theta}_0 + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} G_{i-1}^{\lambda|h+1} + \alpha \boldsymbol{\phi}_h G_h^{\lambda|h+1} \\
&= \mathbf{A}_0^h \boldsymbol{\theta}_0 + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} G_{i-1}^{\lambda|h} + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} \big[G_{i-1}^{\lambda|h+1} - G_{i-1}^{\lambda|h}\big] + \alpha \boldsymbol{\phi}_h G_h^{\lambda|h+1} \\
&= (\mathbf{I} - \alpha \boldsymbol{\phi}_h \boldsymbol{\phi}_h^\top) \Big[\mathbf{A}_0^{h-1}\,\boldsymbol{\theta}_0 + \alpha \sum_{i=1}^{h} \mathbf{A}_i^{h-1}\,\boldsymbol{\phi}_{i-1} G_{i-1}^{\lambda|h}\Big] \\
&\quad + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} \big[G_{i-1}^{\lambda|h+1} - G_{i-1}^{\lambda|h}\big] + \alpha \boldsymbol{\phi}_h G_h^{\lambda|h+1} \\
&= (\mathbf{I} - \alpha \boldsymbol{\phi}_h \boldsymbol{\phi}_h^\top)\,\boldsymbol{\theta}_h + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} \big[G_{i-1}^{\lambda|h+1} - G_{i-1}^{\lambda|h}\big] + \alpha \boldsymbol{\phi}_h G_h^{\lambda|h+1} \\
&= (\mathbf{I} - \alpha \boldsymbol{\phi}_h \boldsymbol{\phi}_h^\top) \boldsymbol{\theta}_h + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h+1-i} \delta'_h + \alpha \boldsymbol{\phi}_h \big[R_{h+1} + \gamma \boldsymbol{\theta}_h{}^\top \boldsymbol{\phi}_{h+1}\big] \\
&= \boldsymbol{\theta}_h + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h+1-i} \delta'_h + \alpha \boldsymbol{\phi}_h \big[R_{h+1} + \gamma \boldsymbol{\theta}_h{}^\top \boldsymbol{\phi}_{h+1} - \boldsymbol{\theta}_h \boldsymbol{\phi}_h\big] \\
&= \boldsymbol{\theta}_h + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h+1-i} \delta'_h \\
&\quad + \alpha \boldsymbol{\phi}_h \big[R_{h+1} + \gamma \boldsymbol{\theta}_h{}^\top \boldsymbol{\phi}_{h+1} - \boldsymbol{\theta}_{h-1} \boldsymbol{\phi}_h + \boldsymbol{\theta}_{h-1} \boldsymbol{\phi}_h - \boldsymbol{\theta}_h \boldsymbol{\phi}_h\big] \\
&= \boldsymbol{\theta}_h + \alpha \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h+1-i} \delta'_h + \alpha \boldsymbol{\phi}_h \delta'_h + \alpha \boldsymbol{\phi}_h \big[\boldsymbol{\theta}_{h-1} \boldsymbol{\phi}_h - \boldsymbol{\theta}_h \boldsymbol{\phi}_h\big] \\
&= \boldsymbol{\theta}_h + \alpha \sum_{i=1}^{h+1} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h+1-i} \delta'_h + \alpha \boldsymbol{\phi}_h \big[\boldsymbol{\theta}_{h-1} \boldsymbol{\phi}_h - \boldsymbol{\theta}_h \boldsymbol{\phi}_h\big] \\
&= \boldsymbol{\theta}_h + \alpha \mathbf{e}_h \delta'_h + \alpha \boldsymbol{\phi}_h \big[\boldsymbol{\theta}_{h-1} \boldsymbol{\phi}_h - \boldsymbol{\theta}_h \boldsymbol{\phi}_h\big] \qquad \text{with } \mathbf{e}_h := \sum_{i=1}^{h+1} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h+1-i} \\
&= \boldsymbol{\theta}_h + \alpha \mathbf{e}_h \big[\delta_h + \boldsymbol{\theta}_h^\top \boldsymbol{\phi}_h - \boldsymbol{\theta}_{h-1}^\top \boldsymbol{\phi}_h\big] + \alpha \boldsymbol{\phi}_h \big[\boldsymbol{\theta}_{h-1}^\top \boldsymbol{\phi}_h - \boldsymbol{\theta}_h^\top \boldsymbol{\phi}_h\big] \\
&= \boldsymbol{\theta}_h + \alpha \mathbf{e}_h \delta_h + \alpha \big[\boldsymbol{\theta}_h^\top \boldsymbol{\phi}_h - \boldsymbol{\theta}_{h-1}^\top \boldsymbol{\phi}_h\big] [\mathbf{e}_h - \boldsymbol{\phi}_h] \qquad (17)
\end{aligned}
$$

We now have the update rule for $\boldsymbol{\theta}_h$, in addition to an explicit definition of $\boldsymbol{e}_h$. Next, using this explicit definition, we derive an update rule to compute $\boldsymbol{e}_h$ from $\boldsymbol{e}_{h-1}$.

$$\begin{aligned}
\boldsymbol{e}_h &= \sum_{i=1}^{h+1} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h+1-i} \\
&= \sum_{i=1}^{h} \mathbf{A}_i^h \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h+1-i} + \boldsymbol{\phi}_h \\
&= (\mathbf{I} - \alpha \boldsymbol{\phi}_h \boldsymbol{\phi}_h^\top) \gamma\lambda \sum_{i=1}^{h} \mathbf{A}_i^{h-1} \boldsymbol{\phi}_{i-1} (\gamma\lambda)^{h-i} + \boldsymbol{\phi}_h \\
&= (\mathbf{I} - \alpha \boldsymbol{\phi}_h \boldsymbol{\phi}_h^\top) \gamma\lambda \boldsymbol{e}_{h-1} + \boldsymbol{\phi}_h \\
&= \gamma\lambda \boldsymbol{e}_{h-1} + \boldsymbol{\phi}_h + \alpha\gamma\lambda (\boldsymbol{e}_{h-1}^\top \boldsymbol{\phi}_h) \boldsymbol{\phi}_h \qquad (18)
\end{aligned}$$

Equations (17) and (18), together with the definition of δ_h, form the true online TD(λ) update equations.

7. Other True Online Methods

In the previous section, we showed that the true online TD(λ) equations can be derived directly from the real-time forward view equations. By using different real-time forward views, new true online methods can be derived. Sometimes, small changes in the real-time forward view, like using a time-dependent step-size, can result in surprising changes in the true online equations. In this section, we look at a number of such variations.

7.1 True Online TD(λ) with Time-Dependent Step-size

When using a time-dependent step-size in the base equation of the forward view (Equation 11) and deriving the update equations following the procedure from Section 6.3, it turns out that a slightly different trace definition appears. We indicate this new trace using a '+' superscript: $\boldsymbol{e}^+$. For fixed step-size, this new trace definition is equal to:

$$\boldsymbol{e}_t^+ = \alpha \boldsymbol{e}_t \,, \qquad \text{for all } t.$$

Of course, using $\boldsymbol{e}_t^+$, instead of $\boldsymbol{e}_t$ also changes the weight vector update slightly. Below, the full set of update equations is shown:

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \\
\boldsymbol{e}_t^+ &= \gamma\lambda \boldsymbol{e}_{t-1}^+ + \alpha_t \boldsymbol{\phi}_t - \alpha_t \gamma\lambda [(\boldsymbol{e}_{t-1}^+)^\top \boldsymbol{\phi}_t] \boldsymbol{\phi}_t \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \delta_t \boldsymbol{e}_t^+ + [\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t][\boldsymbol{e}_t^+ - \alpha_t \boldsymbol{\phi}_t]
\end{aligned}$$

In addition, $\boldsymbol{e}_{-1}^+ := 0$. We can simplify the weight update equation slightly, by using

$$\delta_t' = \delta_t + \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t \,,$$

which changes the update equations to:

$$\begin{aligned}
\delta'_t &= R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t \\
\boldsymbol{e}_t^+ &= \gamma\lambda \boldsymbol{e}_{t-1}^+ + \alpha_t \boldsymbol{\phi}_t - \alpha_t \gamma \lambda [(\boldsymbol{e}_{t-1}^+)^\top \boldsymbol{\phi}_t]\, \boldsymbol{\phi}_t \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \delta'_t\, \boldsymbol{e}_t^+ - \alpha_t [\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t] \boldsymbol{\phi}_t \,.
\end{aligned}$$

Algorithm 2 shows the corresponding pseudocode. Of course, this pseudocode can also be used for constant step-size.

Algorithm 4 true online TD(λ) for time-dependent step-size

INPUT: $\lambda, \boldsymbol{\theta}_{init}$, α_t for $t \geq 0$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init}$, $\hat{v}_{old} \leftarrow 0$, $t \leftarrow 0$
Loop (over episodes):
  obtain initial $\boldsymbol{\phi}$
  $\boldsymbol{e}^+ \leftarrow \mathbf{0}$
  While terminal state is not reached, do:
    obtain next feature vector $\boldsymbol{\phi}'$, γ and reward R
    $\hat{v} \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\phi}$
    $\hat{v}' \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\phi}'$
    $\delta' \leftarrow R + \gamma\, \hat{v}' - \hat{v}_{old}$
    $\boldsymbol{e}^+ \leftarrow \gamma\lambda \boldsymbol{e}^+ + \alpha_t \boldsymbol{\phi} - \alpha_t \gamma \lambda ((\boldsymbol{e}^+)^\top \boldsymbol{\phi})\, \boldsymbol{\phi}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta' \boldsymbol{e}^+ - \alpha_t (\hat{v} - \hat{v}_{old}) \boldsymbol{\phi}$
    $\hat{v}_{old} \leftarrow \hat{v}'$
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi}'$
    $t \leftarrow t + 1$

7.2 True online version of Watkins's Q(λ)

So far, we just considered *on-policy* methods, that is, methods that evaluate a policy that is the same as the policy that generates the samples. However, the true online principle can also be applied to *off-policy* methods, for which the evaluation policy is different from the behaviour policy. As a simple example, consider Watkins's Q(λ) (Watkins, 1989). This is an off-policy method that evaluates the greedy policy given an arbitrary behaviour policy. It does this by combining accumulating traces with a TD error that uses the maximum state-action value of the successor state:

$$\delta_t = R_{t+1} + \max_a \hat{q}(S_t, a) - \hat{q}(S_t, A_t) \,.$$

In addition, traces are reset to 0 whenever a non-greedy action his taken.

From a real-time forward-view perspective, the strategy of Watkins's Q(λ) method can be interpreted as a growing update target that stops growing once a non-greedy action is taken. Specifically, let τ be the first time step *after* time step t that a non-greedy action is taken, then the interim update target for time step t can be defined as:

$$U_t^h := (1-\lambda) \sum_{n=1}^{z-t-1} \lambda^{n-1} G_t^{(n)} \left(\boldsymbol{\theta}_{t+n-1}^{t+n-1} \right) + \lambda^{z-t-1} G_t^{(z-t)} \left(\boldsymbol{\theta}_{z-1}^{z-1} \right), \qquad z = min\{h, \tau\} \,,$$

with

$$G_t^{(n)}(\boldsymbol{\theta}) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \max_a \boldsymbol{\theta}^\top \boldsymbol{\psi}(S_{t+n}, a)\,.$$

Algorithm 5 shows the pseudocode for the true online method that corresponds with this update target definition.

Algorithm 5 true online version of Watkins's Q(λ)

INPUT: $\alpha, \lambda, \gamma, \boldsymbol{\theta}_{init}, \Psi$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init}, \quad \hat{q}_{old} \leftarrow 0$
Loop (over episodes):
  obtain initial state S
  select action A based on state S (for example ϵ-greedy)
  $\boldsymbol{\psi} \leftarrow$ features corresponding to S, A
  $\boldsymbol{e} \leftarrow \mathbf{0}$
  While terminal state has not been reached, do:
    take action A, observe next state S' and reward R
    select action A' based on state S'
    $A^* \leftarrow \text{argmax}_a[\boldsymbol{\theta}^\top \boldsymbol{\psi}(S', a)]$ (if A' ties for the max, then $A^* \leftarrow A'$)
    $\boldsymbol{\psi}' \leftarrow$ features corresponding to S', A^* (if S' is terminal state, $\boldsymbol{\psi}' \leftarrow \mathbf{0}$)
    $\hat{q} \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\psi}$
    $\hat{q}' \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\psi}'$
    $\delta \leftarrow R + \gamma\, \hat{q}' - \hat{q}$
    $\boldsymbol{e} \leftarrow \gamma\lambda \boldsymbol{e} + \boldsymbol{\psi} - \alpha\gamma\lambda[\boldsymbol{e}^\top \boldsymbol{\psi}]\, \boldsymbol{\psi}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\, \boldsymbol{e} + \alpha(\hat{q} - \hat{q}_{old})(\boldsymbol{e} - \boldsymbol{\psi})$
    if $A^* \neq A' : \boldsymbol{e} \leftarrow \mathbf{0}$
    $\hat{q}_{old} \leftarrow \hat{q}'$
    $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi}'\,;\ A \leftarrow A'$

A problem with Watkins's Q(λ) is that if the behaviour policy is very different from the greedy policy, then traces are reset very often, reducing the overall effect of the traces. Sutton et al. (2014) present a more advanced off-policy method based on the true online approach.

7.3 Tabular True Online TD(λ)

Tabular features are a special case of linear function approximation (with one binary feature corresponding to each state). Hence, the update equations for true online TD(λ) that are presented so far also apply to the tabular case. However, we discuss it here separately, because the simplicity of this special case can provide extra insight.

For tabular features, the update equations are:

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}) - \hat{v}(S_t) \\
e_t(s) &= \begin{cases} \gamma\lambda(1-\alpha)e_{t-1}(s) + 1 & \text{if } s = S_t \\ \gamma\lambda e_{t-1}(s) & \text{if } s \neq S_t \end{cases} \\
\hat{v}_{t+1}(s) &= \begin{cases} \hat{v}_t(s) + \alpha\delta_t + \alpha\big[\hat{v}_t(S_t) - \hat{v}_{t-1}(S_t)\big]\,(e_t(s) - 1) & \text{if } s = S_t \\ \hat{v}_t(s) + \alpha\big[\delta_t + \hat{v}_t(S_t) - \hat{v}_{t-1}(S_t)\big]\,e_t(s) & \text{if } s \neq S_t \end{cases}
\end{aligned}$$

What is interesting about the tabular case is that the dutch-trace update reduces to a particularly simple form. In fact, for the tabular case, a dutch-trace update is equal to the weighted average between an accumulating-trace update and a replacing-trace update, with the weight of the former $(1-\alpha)$ and the latter α. Algorithm 6 shows the corresponding pseudocode.

Algorithm 6 tabular true online TD(λ)

```
initialize v(s) for all s
v_old ← 0
Loop (over episodes):
    initialize S
    e(s) ← 0 for all s
    While S is not terminal, do:
        obtain next state S' and reward R
        Δv ← v(S) − v_old
        v_old ← v(S')
        δ ← R + γ v(S') − v(S)
        e(S) ← (1 − α)e(S) + 1
        For all s:
            v(s) ← v(s) + α(δ + Δv)e(s)
            e(s) ← γλe(s)
        v(S) ← v(S) − αΔv
        S ← S'
```

7.4 Non-Linear Function Approximation

An interesting direction for future work is to explore true online methods based on non-linear function approximation. This is especially interesting given the increasing interest in combining reinforcement learning with deep learning (for example, see Mnih et al., 2015). Being able to use higher λ-values reduces the bias of update targets, which moves the policy evaluation task more towards a supervised learning task, on which deep learning excels.

Constructing a real-time forward view for non-linear function approximation is straightforward. The interim λ-return can simply be combined with a non-linear base equation. Let $\hat{v}(s, \boldsymbol{\theta})$ be the value estimate of s given weight vector $\boldsymbol{\theta}$. Then, the following non-linear base equation can be used:

$$\boldsymbol{\theta}^h_{t+1} = \boldsymbol{\theta}^h_t + \alpha\Big[U^h_t - \hat{v}(S_t, \boldsymbol{\theta}^h_t)\Big]\nabla_{\boldsymbol{\theta}}\hat{v}(S_t, \boldsymbol{\theta}^h_t)\,, \tag{19}$$

where $\nabla_{\boldsymbol{\theta}} v(s, \boldsymbol{\theta}_t^h)$ is the gradient of $\hat{v}$ with respect to $\boldsymbol{\theta}$ in point $(S_t, \boldsymbol{\theta}_t^h)$. However, it is an open question whether an efficient backward view can be constructed that computes $\boldsymbol{\theta}_{t+1}^{t+1}$ from $\boldsymbol{\theta}_t^t$.

8. Conclusions

We tested the hypothesis that true online TD(λ) (and true online Sarsa(λ)) dominates TD(λ) (and Sarsa(λ)) with accumulating as well as with replacing traces by performing experiments over a wide range of domains. Our extensive results support this hypothesis. In terms of computational cost, TD(λ) has a slight advantage. In the worst case, true online TD(λ) is twice as expensive. In the typical case of sparse features, it is only fractionally more expensive than TD(λ). Memory requirements are the same. In terms of learning speed, true online TD(λ) was often better, but never worse than TD(λ) with either accumulating or replacing traces, across all domains/representations that we tried. Our analysis showed that especially on domains with relatively low stochasticity and frequent revisits of features a large difference in learning speed can be expected. Furthermore, true online TD(λ) has the advantage over TD(λ) with replacing traces that it can be used with non-binary features, and it has the advantage over TD(λ) with accumulating traces that it is less sensitive with respect to its parameters. Finally, we outlined an approach for deriving new true online methods, based on rewriting the equations of a real-time forward view. This may lead to new, interesting methods in the future.

9. Acknowledgements

The authors thank Hado van Hasselt for extensive discussions leading to the refinement of these ideas. This work was supported by grants from Alberta Innovates Technology Futures and the National Science and Engineering Research Council of Canada. Computing resources were provided by Compute Canada through WestGrid.

Appendix A. Detailed Results Random MRPs

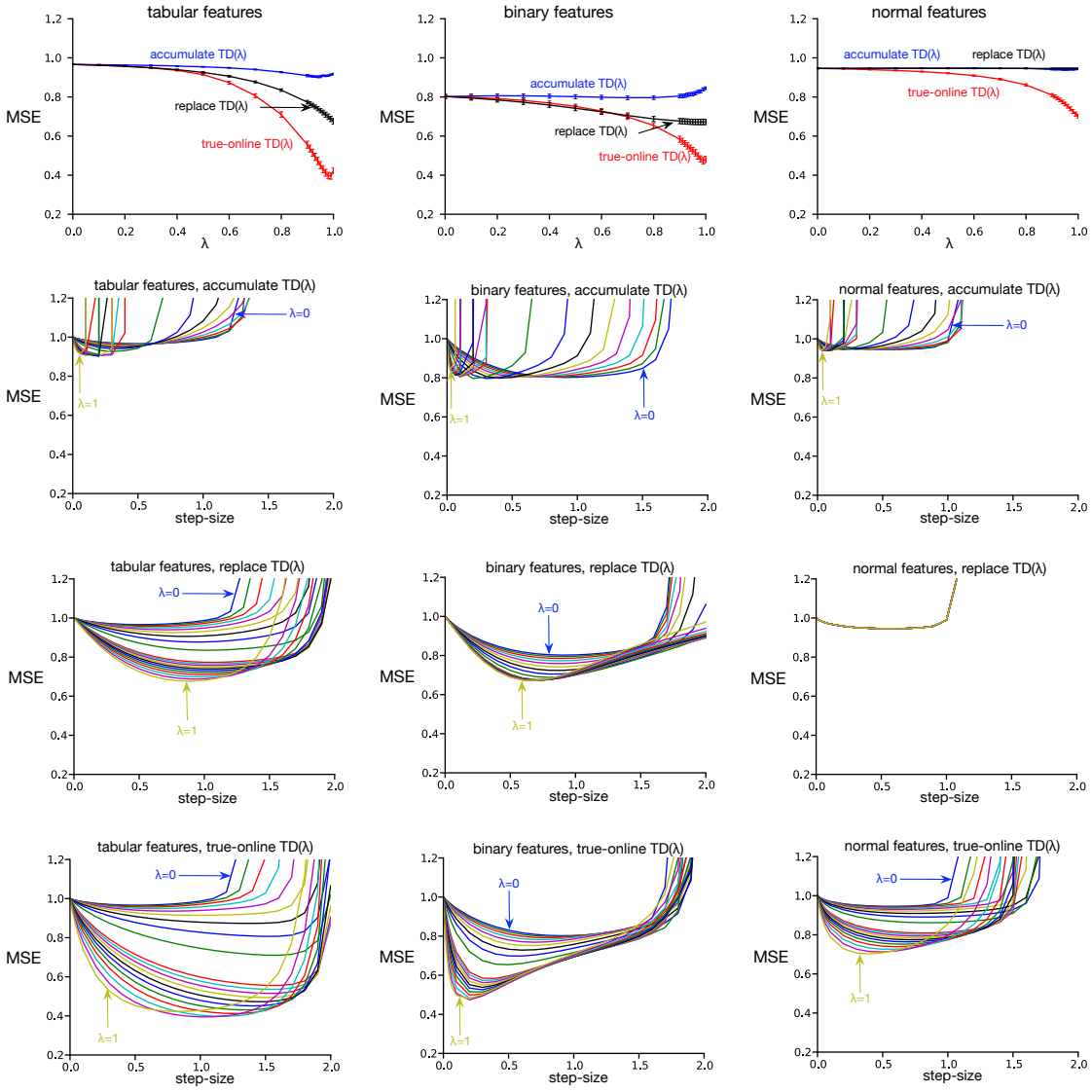

Figure 10: Results on a random MRP with $k = 10$, $b = 3$ and $\sigma = 0.1$. MSE is the mean squared error averaged over the first 100 time steps, as well as 50 runs, and normalized using the initial error. The top graphs summarize the results from the graphs below it; it shows the MSE error, for each λ, at the best step-size.

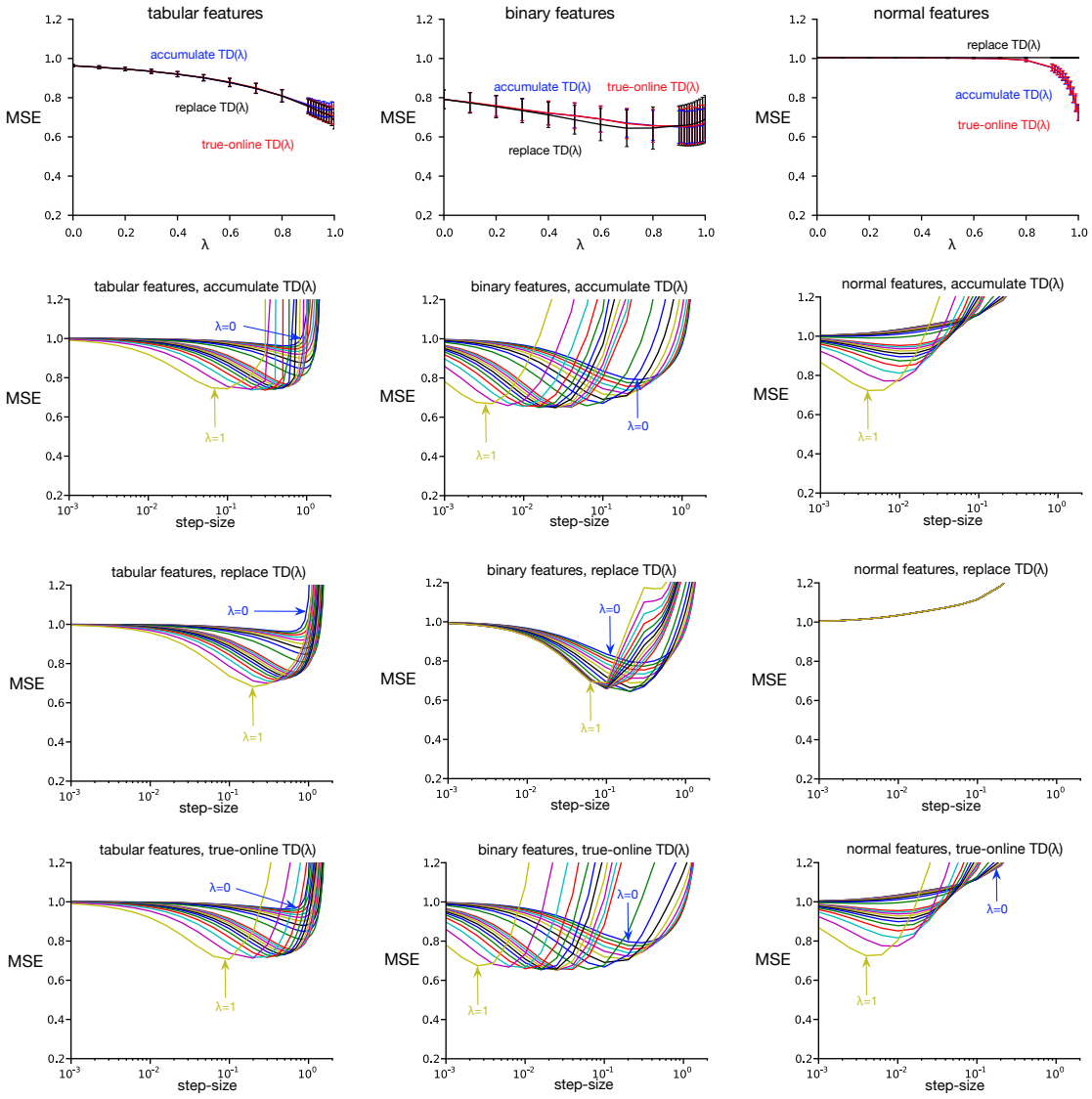

Figure 11: Results on a random MRP with $k = 100$, $b = 10$ and $\sigma = 0.1$. MSE is the mean squared error averaged over the first 1000 time steps, as well as 50 runs, and normalized using the initial error.

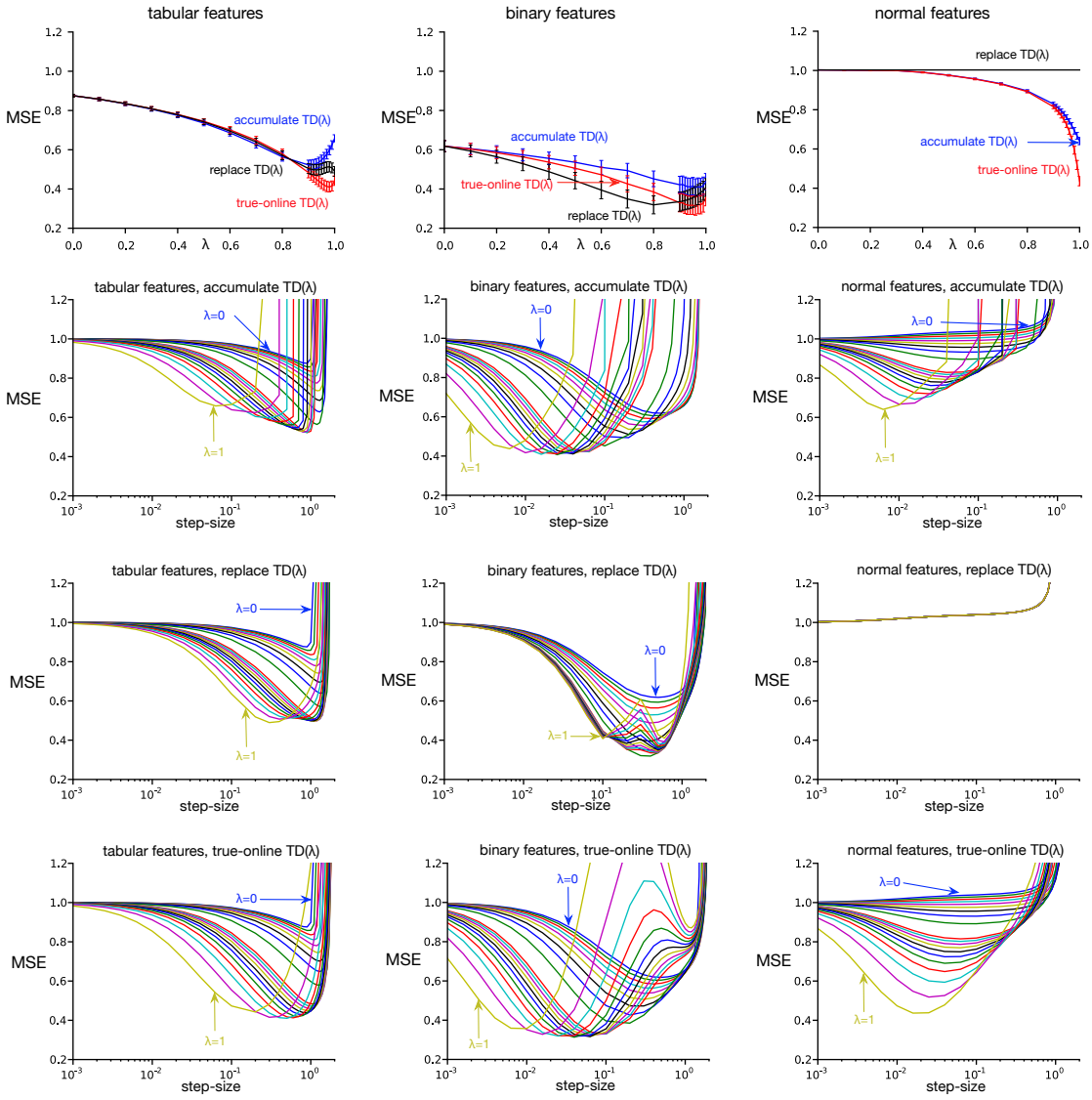

Figure 12: Results on a random MRP with $k = 100$, $b = 3$ and $\sigma = 0$. MSE is the mean squared error averaged over the first 1000 time steps, as well as 50 runs, and normalized using the initial error.

Appendix B. Detailed Results for Myoelectric Prosthetic Arm

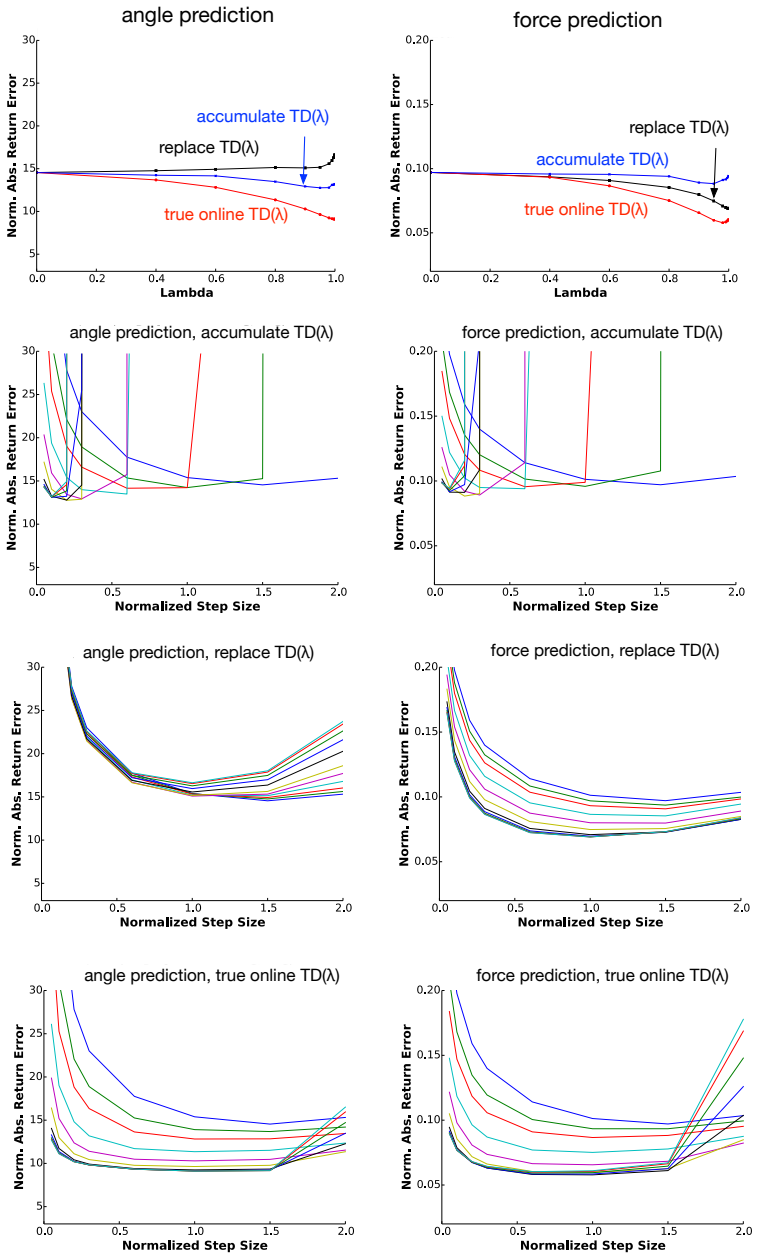

Figure 13: Results on prosthetic data from the single amputee subject described in Pilarski et al. (2013), for the prediction of servo motor angle (*left column*) and grip force (*right column*) as recorded from the amputee's myoelectrically controlled robot arm during a grasping task.

References

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning*, 8(3):341–362.

Defazio, A. and Graepel, T. (2014). A comparison of learning algorithms on the arcade learning environment. In *arXiv:1410.8620*.

Hebert, J. S., Olson, J. L., Morhart, M. J., Dawson, M. R., Marasco, P. D., Kuiken, T. A., and Chan, K. M. (2014). Novel targeted sensory reinnervation technique to restore functional hand sensation after transhumeral amputation. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(4):763–773.

Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., Kumaran, H. K. D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Modayil, J., White, A., and Sutton, R. S. (2014). Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, 22(2):146–160.

Parker, P., Englehart, K. B., , and Hudgins, B. (2006). Myoelectric signal processing for control of powered limb prostheses. *Journal of Electromyography and Kinesiology*, 16(6):541–548.

Pilarski, P. M., Dawson, M. R., Degris, T., Carey, J. P., Chan, K. M., Hebert, J. S., and Sutton, R. S. (2013). Adaptive artificial limbs: A real-time approach to prediction and anticipation. *IEEE Robotics & Automation Magazine*, 20(1):53–64.

Schapire, R. E. and Warmuth, M. K. (1996). On the worst-case analysis of temporal-difference learning algorithms. *Machine Learning*, 22((1/2/3):95–121.

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

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

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

Sutton, R. S., Maei, H. R., and Szepesvári, C. (2009b). A convergent $\mathcal{O}(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. In *Proceedings of Advances in Neural Information Processing Systems 21 (NIPS)*, pages 1609–1616.

Sutton, R. S., Mahmood, A. R., Precup, D., and van Hasselt, H. (2014). A new Q(λ) with interim forward view and Monte Carlo equivalence. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*.

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 unsupervised sensorimotor interaction. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 761–768.

Szepesvári, C. (2010). *Algorithms for reinforcement learning*. Morgan and Claypool.

van Seijen, H. H. and Sutton, R. S. (2014). True online TD(λ). In *Proceedings of the 31th international conference on Machine learning (ICML)*.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England.