True Online TD(λ)

Abstract

TD(λ) is a core algorithm of modern reinforcement learning. The appeal of TD(λ) comes from its clear and conceptually simple forward view, and the fact that it can be implemented online in an inexpensive manner. While offline TD(λ) matches the forward view exactly, online TD(λ) — which is more interesting for practical applications — only approximates it. Up to now, it has been an open question whether a version of online TD(λ) could be made that matches the forward view exactly. We introduce a new online TD(λ) algorithm for function approximation, with the same complexity as the regular version, that achieves the forward view exactly. Key to this is a refinement of the online version of the forward view that is well defined at each time step rather than only at the end of an episode. We use this refined version to derive the new algorithm. By adhering more truly to the goal of matching the forward view, the new algorithm performs much better in practise. We demonstrate this on several standard benchmark problems, where it outperforms both accumulating and replacing traces.

1. Why True Online TD(λ) Matters

Temporal-difference (TD) learning is a core learning technique in modern reinforcement learning (Sutton, 1988; Kaelbling et al., 1996; Sutton & 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) 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 the standard implementation of TD(λ), which uses *accumulating traces*, interprets the updates under TD(λ) as updates with the *λ-return* as update target. The λ-return is an estimate of the expected return based on rewards as well as other expected-return estimates, with λ determining the exact way they are combined.

In the *offline* case, where the expected-return estimates are updated after all data has been collected, the equivalence between TD(λ) and the forward view is exact. However, in the more relevant *online* case, where estimates are updated *during* data collection, the equivalence only holds approximately. Because the equivalence does not hold exactly, it is possible that estimates diverge under online TD(λ), even for simple tasks with bounded returns (we show an example in Section 4.3).

To avoid divergence of estimates, an alternative implementation of TD(λ) is sometimes used, based on *replacing traces* (Singh et al., 1996), which prevents such behaviour. However, this solution is far from ideal. While it can prevent divergence of estimates, this can come at the cost of losing all learning speed benefits of eligibility traces. In particular, when a large fraction of the features have a non-zero value, replacing traces looses its effectiveness (we show an example in Section 4.3.). In addition, in the control case, where expected-return estimates are learned conditioned on actions, there are two ways to implement replacing traces (with or without clearing traces for non-selected actions), while it is unclear which implementation is best, adding an extra parameter.

In this paper, we present for the first time (to the best our knowledge) an online version of TD(λ) that is exactly equivalent to the forward view. That is, the values computed by this version of TD(λ), which we call *true online TD(λ)*, are exactly the same as the values computed by the forward view, *at every moment*

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

in time. As a consequence, true online TD(λ) avoids divergence of estimates that can occur under accumulating traces, while it does not suffer from the negative side-effects of replacing traces. We demonstrate this empirically, by showing that true online TD(λ) is able to perform well on a benchmark problem where both accumulating and replacing traces are ineffective.

Key to our new implementation of TD(λ) is a refinement of the forward view. The current forward view is not well-defined in the online case, because its update target assumes knowledge of rewards and states that are not yet observed at the moment of the update. We refined the forward view, such that it is well-defined in the online case at all time steps.

Next, we present the main learning framework that we consider in this paper.

2. Problem Setting and Notation

In this section, we present the main learning framework that we consider and discuss TD(λ) and its forward view. 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 small letters (e.g., v), and sets by calligraphic font (e.g., $\mathcal{S}$, $\mathcal{A}$).

2.1. Markov Reward Processes

We focus in this paper primarily on discrete-time *Markov reward processes* (MRPs), which can be described as 4-tuples of the form $\langle \mathcal{S}, p, r, \gamma \rangle$, consisting of $\mathcal{S}$, the set of all states; $p(s'|s)$, the transition probability function, giving for each state $s \in \mathcal{S}$ the probability of a transition to state $s' \in \mathcal{S}$ at the next step; $r(s, s')$, the reward function, giving the expected reward after a transition from s to s'. γ is the discount factor, specifying how future rewards are weighted with respect to the immediate reward.

The *return* at time step t is the discounted sum of rewards observed after time step t:

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

An MRP can contain *terminal states*, dividing 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 for an episodic MRP is defined as:

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

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

We are interested in learning the value-function v of an MRP, which maps each state $s \in \mathcal{S}$ to the expected value of the return:

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

Next, we discuss function approximation.

2.2. Function Approximation

In the general case, the state-space can be huge or even continuous. In such cases it is not possible to represent the value function v exactly. In this case, or when data-generalization is required, function approximation is used to represent v. The approximate value-function $\hat{v}(s, \boldsymbol{\theta}_t)$ gives the approximate value of state s, given the weight vector $\boldsymbol{\theta}$ at time step t. As a shorthand, we indicate this value by $\hat{v}_t(s)$.

A common approach is to use linear function approximation, in which case the value of a state is the inner product between the weight vector $\boldsymbol{\theta}$ and a feature vector $\boldsymbol{\phi}$. In this case, the value of state s at time step t is approximated by:

$$\hat{v}_t(s) = \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(s) = \sum_i \theta_{i,t} \, \phi_i(s)$$

where $\phi_i(s)$ is the value of feature i in state s, and $\theta_{i,t}$ is the weight of that feature at time step t.

One of the main technique for improving the value function estimate $\hat{v}$ is by means of gradient descent, which involves incremental adjustments of the weight vector $\boldsymbol{\theta}$ in the direction of the gradient:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \, [U_t - \hat{v}_t(S_t)] \nabla_{\boldsymbol{\theta}_t} \hat{v}_t(S_t) \,, \qquad (1)$$

where $\nabla_{\boldsymbol{\theta}_t} \hat{v}_t(S_t)$ is the gradient of $\hat{v}$ with respect to $\boldsymbol{\theta}_t$. In case of linear function approximation, this gradient has a particularly simple form:

$$\nabla_{\boldsymbol{\theta}_t} \hat{v}_t(s) = \boldsymbol{\phi}(s) \,.$$

There are many ways to construct an update target U_t from observed states and rewards. For example, *Monte Carlo* methods use the full return as the update target:

$$U_t = G_t \,.$$

TD methods use an update target that is based on value estimates of other states. For example, the TD(0) update target is:

$$U_t = R_{t+1} + \gamma \hat{v}_t(S_{t+1}) \,. \qquad (2)$$

The update shown in Equation (1) is an *online* update, referring to the fact that the weight vector is updated at every time step t. Alternatively, the weight vector can be updated *offline*. With offline updating, the weight vector stays constant during an episode, and instead the weight corrections are collected on the side. Let the weight vector for episode k be $\boldsymbol{\theta}_k$. The weight correction for time step t of this episode is:

$$\boldsymbol{\Delta}_t = \alpha \left[U_t - \hat{v}_k(S_t)\right] \nabla_{\boldsymbol{\theta}_k} \hat{v}_k(S_t) \,.$$

After the episode has terminated, the weight vector is updated by adding all weight corrections collected during the episode:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sum_{t=0}^{T-1} \boldsymbol{\Delta}_t \,.$$

Online updating not only has the advantage that it can be applied to non-episodic tasks, but it will in general also produce better value-function estimates under temporal-difference learning. The reason is that under online learning the update targets, which bootstrap from the values of other states, use more recent value estimates.

2.3. TD(λ)

Under linear function approximation, the TD(0) method, based on update target (2), performs at time step $t+1$ the following update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\, \delta_t \boldsymbol{\phi}(S_t)$$

where

$$\delta_t = R_{t+1} + \gamma \hat{v}_t(S_{t+1}) - \hat{v}_t(S_t)$$

is called the *TD error*. This update only affects the weights θ_i for which ϕ_i is non-zero. The idea behind TD(λ) is to update weights for which ϕ_i was non-zero in the (near) past as well. This is implemented by means of an eligibility vector $\boldsymbol{e}$, which reflects how much each feature is 'eligible' for the current TD error. TD(λ) performs an update of each θ_i, with the current TD error, proportional to its trace value e_i:[1]

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \delta_t \boldsymbol{e}_t \,.$$

For the standard implementation, which uses *accumulating traces*, $\boldsymbol{e}_t$ is initialized as the zero vector, $\mathbf{0}$, and updated according to:

$$\boldsymbol{e}_t = \gamma\lambda \boldsymbol{e}_{t-1} + \alpha\, \boldsymbol{\phi}(S_t) \,,$$

where λ is the trace-decay parameter. Note that for $\lambda = 0$, the TD(λ) update reduces to the $TD(0)$ update. Algorithm 1 shows pseudocode for TD(λ).

[1] We fold the step-size α in the eligibility vector.

Algorithm 1 linear TD(λ)

```
initialize θ arbitrarily
loop {over episodes}
  initialize e = 0
  initialize S
  repeat {for each step in the episode}
    generate reward R and next state S' for S
    δ ← R + γ θ^⊤ φ(S') − θ^⊤ φ(S)
    e ← γλe + αφ(S)
    θ ← θ + δe
    S ← S'
  until S is terminal
end loop
```

It is well-known that for TD(λ) with accumulating traces values sometimes diverge, even on simple tasks with bounded returns. For this reason, a second implementation of TD(λ) is sometimes used, based on *replacing traces* (Singh et al., 1996). With replacing traces, $\boldsymbol{e}$ is updated as follows:

$$e_{i,t} = \begin{cases} \gamma\lambda e_{i,t-1} & \text{if } \phi_i(S_t) = 0 \\ \alpha\phi_i(S_t) & \text{if } \phi_i(S_t) \neq 0 \end{cases} \qquad \text{for all } i \,.$$

While replacing traces prevents divergence of values, its application is limited. The fundamental limitation of replacing traces is that it treats zero as a special case. When there are only a small number of features with a value of zero, TD(λ) with replacing traces approaches TD(0), even for high λ values. If all features are non-zero, $\boldsymbol{e}_t = \alpha\, \boldsymbol{\phi}(S_t)$ and the TD(λ) replacing-traces update is equal to the TD(0) update.

2.4. The Forward View

The forward view relates TD(λ) (with accumulating traces) to the λ-*return algorithm*. This algorithm performs at each time step a standard update (as in Equation (1)) with the λ-*return* as update target. The λ-return G_t^λ is an estimate of the expected return based on a combinations of rewards and other value estimates:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \,,$$

with $G_t^{(n)}$ the n-step return:

$$G_t^{(n)} = \sum_{i=1}^{n} \gamma^{i-1} R_{t+i} + \gamma^n\, \hat{v}_t(S_{t+n}) \,.$$

For an episodic task, the λ-return is defined as:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t^{(T-t)} \,, \quad (3)$$

where T is the time step that the terminal state is reached. Because the value of a terminal state is always 0, the last λ-return in this equation is equal to the full return:

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

Note that for $\lambda = 0$, G_t^λ is equal to $R_{t+1} + \gamma \hat{v}_t(S_{t+1})$, the TD(0) update target. On the other hand, for $\lambda = 1$, G_t^λ is equal to the full return G_t. For λ in between 0 and 1, some mix between the TD(0) update target and the Monte Carlo update target is obtained.

It has been shown that offline TD(λ) is equal to the offline λ-return algorithm (Sutton, 1988; Sutton & Barto, 1998). However, online TD(λ) is only approximately equal to the online λ-return algorithm. In fact, no version of online TD(λ) can be constructed that matches the online λ-return algorithm exactly, because the λ-return uses rewards and states beyond the current time step. Therefore, in the next section, we refine the forward view such that it can be implemented online.

3. True Online Forward View

The problem with the current forward view is that it cannot be implemented online, because the update at times step $t+1$, based on λ-return G_t^λ, uses rewards and states beyond time step $t+1$ (in fact, potentially from infinitely far in the future). Hence, in order to make an online TD(λ) version that is truly equivalent to the forward view, the forward view should be extended such that it has a well-defined, implementable, online version.

To construct an implementable forward view, we start by generalizing the λ-return to a version that is truncated at a specific time step. Let t be the time step the λ-return is truncated and $\tau < t$ be the time step from which the λ-return starts. The *truncated λ-return* $G_{\tau,t}^\lambda$ is defined as:

$$G_{\tau,t}^\lambda = (1-\lambda) \sum_{n=1}^{t-\tau-1} \lambda^{n-1} G_{\tau,\tau'(n)}^{(n)} + \lambda^{t-\tau-1} G_{\tau,\tau'(t-\tau)}^{(t-\tau)} \,.$$

Note that the n-step returns in this definition also have a second time-step subscript: $\tau'(n)$. This time step refers to the time step of the state value used in the n-step return:

$$G_{\tau,\tau'}^{(n)} = \sum_{i=1}^{n} \gamma^{i-1} R_{\tau+i} + \gamma^n \, \hat{v}_{\tau'}(S_{\tau+n}) \qquad (4)$$

Having this second time step allows us to define n-step returns that use more recent value estimate than the one from time step τ (note: $G_{\tau,\tau}^{(n)} = G_\tau^{(n)}$). For an episodic task terminating at time step T, $G_{\tau,T}^\lambda$ is equal to the regular λ-return G_τ^λ (see Equation 3) if $\tau'(n) = \tau$ for $1 \leq n \leq t - \tau$. We come back to the specific definition of $\tau'(n)$ at the end of this section.

The idea behind the new online forward view is in essence a simple one: at each time step, the truncated λ-returns from all previous time steps are updated, such that they are now truncated at the current time step; the weight vector of the current time step is determined by sequentially performing TD backups, using the updated λ-returns, starting from the initial weight vector, $\boldsymbol{\theta}_{init}$. The weight vectors for the first three time steps are shown below. We use a second index for $\boldsymbol{\theta}$ to indicate at which time step the λ-returns that are used to construct it are truncated.[2]

$$\begin{aligned}
\boldsymbol{\theta}_{0,0}: \quad & \boldsymbol{\theta}_{0,0} = \boldsymbol{\theta}_{init} \,, \\
\\
\boldsymbol{\theta}_{1,1}: \quad & \boldsymbol{\theta}_{0,1} = \boldsymbol{\theta}_{init} \\
& \boldsymbol{\theta}_{1,1} = \boldsymbol{\theta}_{0,1} + \alpha_0 \Big[G_{0,1}^\lambda - \boldsymbol{\theta}_{0,1}^\top \boldsymbol{\phi}(S_0) \Big] \boldsymbol{\phi}(S_0) \,, \\
\\
\boldsymbol{\theta}_{2,2}: \quad & \boldsymbol{\theta}_{0,2} = \boldsymbol{\theta}_{init} \\
& \boldsymbol{\theta}_{1,2} = \boldsymbol{\theta}_{0,2} + \alpha_0 \Big[G_{0,2}^\lambda - \boldsymbol{\theta}_{0,2}^\top \boldsymbol{\phi}(S_0) \Big] \boldsymbol{\phi}(S_0) \,, \\
& \boldsymbol{\theta}_{2,2} = \boldsymbol{\theta}_{1,2} + \alpha_1 \Big[G_{1,2}^\lambda - \boldsymbol{\theta}_{1,2}^\top \boldsymbol{\phi}(S_1) \Big] \boldsymbol{\phi}(S_1) \,, \\
\\
\boldsymbol{\theta}_{3,3}: \quad & \boldsymbol{\theta}_{0,3} = \boldsymbol{\theta}_{init} \\
& \boldsymbol{\theta}_{1,3} = \boldsymbol{\theta}_{0,3} + \alpha_0 \Big[G_{0,3}^\lambda - \boldsymbol{\theta}_{0,3}^\top \boldsymbol{\phi}(S_0) \Big] \boldsymbol{\phi}(S_0) \,, \\
& \boldsymbol{\theta}_{2,3} = \boldsymbol{\theta}_{1,3} + \alpha_1 \Big[G_{1,3}^\lambda - \boldsymbol{\theta}_{1,3}^\top \boldsymbol{\phi}(S_1) \Big] \boldsymbol{\phi}(S_1) \,, \\
& \boldsymbol{\theta}_{3,3} = \boldsymbol{\theta}_{2,3} + \alpha_2 \Big[G_{2,3}^\lambda - \boldsymbol{\theta}_{2,3}^\top \boldsymbol{\phi}(S_2) \Big] \boldsymbol{\phi}(S_2) \,,
\end{aligned}$$

More generally, the weight vector at time step t, $\boldsymbol{\theta}_{t,t}$, is:

$$\begin{aligned}
\boldsymbol{\theta}_{0,t} &= \boldsymbol{\theta}_{init} \\
\boldsymbol{\theta}_{1,t} &= \boldsymbol{\theta}_{0,t} \\
&\quad + \alpha_0 \Big[G_{0,t}^\lambda - (\boldsymbol{\theta}_{0,t})^\top \boldsymbol{\phi}(S_0) \Big] \boldsymbol{\phi}(S_0) \\
&\vdots \\
\boldsymbol{\theta}_{t,t} &= \boldsymbol{\theta}_{t-1,t} \\
&\quad + \alpha_{t-1} \Big[G_{t-1,t}^\lambda - (\boldsymbol{\theta}_{t-1,t})^\top \boldsymbol{\phi}(S_{t-1}) \Big] \boldsymbol{\phi}(S_{t-1}) \,.
\end{aligned}$$

[2] For ease of exposition, we focus on the linear case. However, our approach can be easily extended to the non-linear case.

Note that $\boldsymbol{\theta}_{i,j}$ for $i \neq j$ are temporary 'helper' vectors. The value estimate of state s at time step t is determined by $\theta_{t,t}$:

$$\hat{v}_t(s) = \boldsymbol{\theta}_{t,t}\boldsymbol{\phi}(s)\,.$$

Similarly, $\hat{v}_{\tau_n}$, used by the the n-step return in Equation 4, makes use of $\boldsymbol{\theta}_{\tau_n,\tau_n}$.

So far, we have not yet specified $\tau'(n)$, used by the truncated λ-return $G^{\lambda}_{\tau,t}$. It might seem logical to define $\tau'(n) = \tau$ for all n, such that $G^{(n)}_{\tau,\tau'(n)}$ reduces to the standard n-step return $G^{(n)}_{\tau}$. However, this would not only reduce the accuracy (older value estimates are used), it is also highly impractical: the $\boldsymbol{\theta}$ vectors from all previous time steps would have to be stored. Instead, we use

$$\tau'(n) = \tau + n - 1\,, \qquad \text{for } 1 \leq n \leq t - \tau\,. \tag{5}$$

With this definition each new n-step return uses the most recent $\boldsymbol{\theta}$ vector, and enables efficient implementation (as we demonstrate in the next section).

We call the algorithm that computes $\boldsymbol{\theta}_{t,t}$ at each time step t the *truncated λ-return algorithm*. Note that at the end of an episode, the values computed by the truncated λ-return algorithm are the same as the values computed by the online λ-return algorithm. However, the values *during* an episode are different. Because of this, the truncated λ-return algorithm can be implemented fully online, in contrast to the λ-return algorithm.

While it is possible to implement the truncated λ-return algorithm online, it is an expensive method and requires storage of all observed states and rewards. In the next section, we introduce true online TD(λ) which implements the true online forward view efficiently using eligibility traces.

4. True Online TD(λ)

This section present the online TD(λ) method that matched the new online forward view exactly. First the algorithm itself is presented, then its equivalence to the truncated λ-return algorithm is proven. The section finishes with empirical results demonstrating the benefits of the algorithm.

4.1. The Algorithm

True online TD(λ) forms the backward view of the truncated λ-return algorithm. Like traditional TD(λ), it updates feature weights proportional to a decaying eligibility trace.

We start by specifying the very first backup. All traces have a value of 0 at this point. Therefore, the backup is simply a TD(0) update:

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \alpha_0[R_1 + \gamma\boldsymbol{\theta}_0^{\top}\boldsymbol{\phi}(S_1) - \boldsymbol{\theta}_0^{\top}\boldsymbol{\phi}(S_0)]\boldsymbol{\phi}(S_0)$$

For $t \geq 1$, $\boldsymbol{\theta}$ receives a TD(0)-like update (note the subtle difference in the weight vector used for S_{t+1} and S_t), preceded by an update of $\boldsymbol{\theta}$ proportional to the trace value at the previous time step, decayed by $\gamma\lambda$:

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

with

$$\delta_t = R_{t+1} + \boldsymbol{\theta}_t^{\top}\boldsymbol{\phi}(S_{t+1}) - \boldsymbol{\theta}_{t-1}^{\top}\boldsymbol{\phi}(S_t)\,. \tag{6}$$

Note that δ_t makes use of $\hat{v}_{t-1}(S_t)$, instead of $\hat{v}_t(S_t)$, as with traditional TD(λ).

Substituting the expression for $\boldsymbol{\theta}'_t$ in the expression for $\boldsymbol{\theta}_{t+1}$ gives:

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

with

$$\boldsymbol{e}_t = \gamma\lambda\boldsymbol{e}_{t-1} + \alpha_t\boldsymbol{\phi}(S_t) - \alpha_t\gamma\lambda[\boldsymbol{e}_{t-1}^{\top}\boldsymbol{\phi}(S_t)]\,\boldsymbol{\phi}(S_t) \tag{7}$$

The equations above define how $\boldsymbol{\theta}$ is updated for true online TD(λ). Algorithm 2 shows pseudo-code that implements these updates.

4.2. Equivalence with Online Forward View

In this section, we prove that the values computed by true online TD(λ) are exactly the same as those computed by the truncated λ-return algorithm. That is, we prove that $\boldsymbol{\theta}_{t,t} = \boldsymbol{\theta}_t$ for all t, where $\boldsymbol{\theta}_{t,t}$ are the weights computed by the truncated λ-return algorithm, and $\boldsymbol{\theta}_t$ are the weights computed by the true online TD(λ) algorithm.

That $\theta_{1,1} = \theta_1$ is true can be easily checked. We now prove that if

$$\boldsymbol{\theta}_{\tau,\tau} = \boldsymbol{\theta}_{\tau-1,\tau-1} + \delta_{\tau-1}\,\boldsymbol{e}_{\tau-1} + \epsilon_{\tau-1}\boldsymbol{\phi}(S_{\tau-1}) \tag{8}$$

Algorithm 2 true online TD(λ)

initialize $\boldsymbol{\theta}$ arbitrarily
loop {over episodes}
  initialize $\boldsymbol{e} = \mathbf{0}$
  initialize S
  $\hat{v}_S \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\phi}(S)$
  repeat {for each step in the episode}
    generate reward R and next state S' for S
    $\hat{v}_{S'} \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\phi}(S')$
    $\delta \leftarrow R + \gamma \hat{v}_{S'} - \hat{v}_S$
    $\boldsymbol{e} \leftarrow \gamma\lambda \boldsymbol{e} + \alpha\big[1 - \gamma\lambda\, \boldsymbol{e}^\top \boldsymbol{\phi}(S)\big]\, \boldsymbol{\phi}(S)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta \boldsymbol{e} + \alpha\big[\hat{v}_S - \boldsymbol{\theta}^\top \boldsymbol{\phi}(S)\big]\, \boldsymbol{\phi}(S)$
    $\hat{v}_S \leftarrow \hat{v}_{S'}$
    $S \leftarrow S'$
  until S is terminal
end loop

with

$$\epsilon_{\tau-1} = \alpha_{\tau-1}[\boldsymbol{\theta}^\top_{\tau-2,\tau-2}\boldsymbol{\phi}(S_{\tau-1}) - \boldsymbol{\theta}^\top_{\tau-1,\tau-1}\boldsymbol{\phi}(S_{\tau-1})]$$

and

$$\delta_{\tau-1} = R_\tau + \boldsymbol{\theta}^\top_{\tau-1,\tau-1}\boldsymbol{\phi}(S_\tau) - \boldsymbol{\theta}^\top_{\tau-2,\tau-2}\boldsymbol{\phi}(S_{\tau-1})\,,$$

and $\boldsymbol{e}_\tau$ given by (7) holds for all $\tau \leq t$, then it holds for $\tau = t+1$. Note that this is sufficient to prove that $\boldsymbol{\theta}_{t,t} = \boldsymbol{\theta}_t$ holds for all t.

By substituting the helper $\boldsymbol{\theta}$ vectors in the expression for $\boldsymbol{\theta}_{t,t}$ given in Section 3, the recursive definition of $\boldsymbol{\theta}_{t,t}$ can be rewritten to a single expression of the form:

$$\boldsymbol{\theta}_{t,t} = \boldsymbol{c}_0 + \boldsymbol{c}_1 G^\lambda_{0,t} + \boldsymbol{c}_2 G^\lambda_{1,t} + ... + \boldsymbol{c}_t G^\lambda_{t-1,t}\,, \qquad (9)$$

with the $\boldsymbol{c}_i$ vectors are constructed from the vectors $\boldsymbol{\theta}_{init}$, $\boldsymbol{\phi}(S_0)$ through $\boldsymbol{\phi}(S_{t-1})$ and step-sizes α_0 through α_{t-1}. In other words, the $\boldsymbol{c}_i$ vectors contain no weight values or rewards. Similarly, the (helper) vector $\boldsymbol{\theta}_{t,t+1}$ can be rewritten as

$$\boldsymbol{\theta}_{t,t+1} = \boldsymbol{c}_0 + \boldsymbol{c}_1 G^\lambda_{0,t+1} + \boldsymbol{c}_2 G^\lambda_{1,t+1} + ... + \boldsymbol{c}_t G^\lambda_{t-1,t+1}\,.$$

The $\boldsymbol{c}_i$ vectors in this equation are the same vectors as those in Equation (9). To compute $\boldsymbol{\theta}_{t,t+1}$ from $\boldsymbol{\theta}_{t,t}$ each involved λ-return should be truncated one time step later.

To see what happens when a truncated λ-return is extended by one time step, consider $G^\lambda_{\tau,t+1} - G^\lambda_{\tau,t}$. $G^\lambda_{\tau,t}$ is defined as (using (5)):

$$G^\lambda_{\tau,t} = (1-\lambda) \sum_{i=1}^{t-\tau-1} \lambda^{n-1} G^{(n)}_{\tau,\tau+n-1} + \lambda^{t-\tau-1} G^{(t-\tau)}_{\tau,t-1},$$

and $G^\lambda_{\tau,t+1}$ is:

$$G^\lambda_{\tau,t+1} = (1-\lambda) \sum_{i=1}^{t-\tau} \lambda^{n-1} G^{(n)}_{\tau,\tau+n-1} + \lambda^{t-\tau} G^{(t+1-\tau)}_{\tau,t},$$

Subtracting $G^\lambda_{\tau,t}$ from $G^\lambda_{\tau,t+1}$ yields

$$G^\lambda_{\tau,t+1} - G^\lambda_{\tau,t} = \lambda^{t-\tau}[G^{(t+1-\tau)}_{\tau,t} - G^{(t-\tau)}_{\tau,t-1}]\,. \qquad (10)$$

Similarly, it can be shown that subtracting $\hat{G}^{(t-\tau)}_{\tau,t}$ from $\hat{G}^{(t+1-\tau)}_{\tau,t-1}$ yields:

$$\begin{aligned}\hat{G}^{(t+1-\tau)}_{\tau,t} - \hat{G}^{(t-\tau)}_{\tau,t-1} &= \gamma^{t-\tau} R_{t+1} + \gamma^{t+1-\tau}\hat{v}_t(S_{t+1}) \\ &\quad - \gamma^{t-\tau}\hat{v}_{t-1}(S_t) \\ &= \gamma^{t-\tau}\delta_t\,.\end{aligned}$$

Substituting this result in (10) yields:

$$G^\lambda_{\tau,t+1} - G^\lambda_{\tau,t} = (\lambda\gamma)^{t-\tau}\delta_t\,. \qquad (11)$$

$G^\lambda_{\tau,t}$ is a value constructed from a weighted sum of rewards and different state values, ending with a term involving the value of S_t:

$$G^\lambda_{\tau,t} = \cdots + \lambda^{t-\tau-1}\gamma^{t-\tau}\hat{v}_{t-1}(S_t)$$

From this and (11) it follows that:

$$\begin{aligned}G^\lambda_{\tau,t+1} &= G^\lambda_{\tau,t} + (\lambda\gamma)^{t-\tau}\delta_t\,. \\ &\quad \cdots + \lambda^{t-\tau-1}\gamma^{t-\tau}\hat{v}_{t-1}(S_t) + (\lambda\gamma)^{t-\tau}\delta_t \\ &\quad \cdots + \lambda^{t-\tau-1}\gamma^{t-\tau}[\hat{v}_{t-1}(S_t) + \lambda\delta_t]\end{aligned}$$

This demonstrates that increasing the truncated λ-return $G^\lambda_{\tau,t}$ by one time step simply means submitting the value $\hat{v}_{t-1}(S_t)$ with the value $\hat{v}_{t-1}(S_t) + \lambda\delta_t$.

Each truncated λ-return in Equation (9) ends with a term containing $\hat{v}_{t-1}(S_t)$ in it (because they are all truncated at time step t). Because we assume (8) holds for $\tau = t$, we have a simple expression that shows the relation between $\boldsymbol{\theta}_{t,t}$ and $\hat{v}_{t-1}(S_t)$:

$$\begin{aligned}\boldsymbol{\theta}_{t,t} &= \boldsymbol{\theta}_{t-1,t-1} + \boldsymbol{e}_{t-1}\delta_{t-1} + \epsilon_{t-1}\boldsymbol{\phi}(S_{t-1}) \\ &= \boldsymbol{\theta}_{t-1,t-1} + \epsilon_{t-1}\boldsymbol{\phi}(S_{t-1}) \\ &\quad + \boldsymbol{e}_{t-1}[R_t + \gamma\hat{v}_{t-1}(S_t) - \hat{v}_{t-2}(S_{t-1})]\end{aligned}$$

Replacing $\hat{v}_{t-1}(s_t)$ by $v_{t-1}(s_t) + \lambda\delta_t$ in the right-hand side of this equation gives $\boldsymbol{\theta}_{t,t+1}$. Hence:

$$\boldsymbol{\theta}_{t,t+1} = \boldsymbol{\theta}_{t,t} + \gamma\lambda\boldsymbol{e}_{t-1}\delta_t\,. \qquad (12)$$

From Section 3 it follows that $\boldsymbol{\theta}_{t+1,t+1}$ is computed from $\boldsymbol{\theta}_{t,t+1}$ as follows:

$$\begin{aligned}\boldsymbol{\theta}_{t+1,t+1} &= \boldsymbol{\theta}_{t,t+1} \\ &\quad + \alpha_t \Big[G^{\lambda}_{t,t+1} - (\boldsymbol{\theta}_{t,t+1})^\top \boldsymbol{\phi}(S_t) \Big] \boldsymbol{\phi}(S_t) \,. \\ &= \boldsymbol{\theta}_{t,t+1} + \alpha_t \big[R_{t+1} + \gamma \hat{v}_t(S_t) \\ &\quad - (\boldsymbol{\theta}_{t,t+1})^\top \boldsymbol{\phi}(S_t) \Big] \boldsymbol{\phi}(S_t) \qquad (13)\end{aligned}$$

By following the same derivation used in Section 4.1 to derive Equation (7), it follows that by substituting (12) in (13) yields Equation (8) with $\tau = t+1$.

4.3. Empirical Results

We compare the performance of true online TD versus traditional TD with accumulating and replacing traces on a random walk task. The random walk task is shown in Figure 1 for $N = 6$ (N being the total number of states, including the terminal state). In our experiment we use $N = 11$. p, the transition probability in the direction of the terminal state, is set to 0.9. Initial $\boldsymbol{\theta}$ is $\mathbf{0}$.

We used linear function approximation with two types of features, resulting in two different tasks. The feature values of these two tasks are shown in Table 1 (for $N = 6$). In task 1, there are (at most) 3 non-zero features for each state. In task 2, the number of non-zero features is between 1 and $N-1$. For the terminal state all features have value 0. The L^2-norm of $\boldsymbol{\phi}(s)$ is 1 for all non-terminal states.

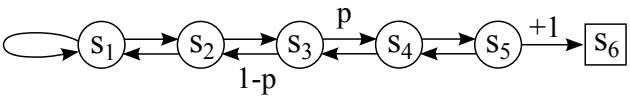

Figure 1. Random walk for $N = 6$. Transition probability to the right is p; transition probability to the left is $1-p$. All rewards are 0, except transition to the terminal state (s_6), which results in a reward of +1. The initial state is s_1.

Figure 2 shows the performance on both tasks for α from 0 to 1 with steps of 0.01 and λ from 0 to 0.9 with steps of 0.1 and from 0.9 to 1.0 with steps of 0.25. Figure 3 shows the performance for the different λ values at the best α value.

While the return has an upper bound of 1, accumulating traces get errors above 1 on both tasks, indicating divergence of values. While replacing traces does not result in divergence of values, its has no effect in the second task. True TD is the only method that achieves a performance benefit (with respect to TD(0)) on both tasks, clearly demonstrating the strength of true online TD(λ).

Table 1. Feature values for random walk task 1 and task 2 for $N = 6$.

		s_1	s_2	s_3	s_4	s_5	s_6
Task 1	ϕ_1	1	$1/\sqrt{2}$	$1/\sqrt{3}$	0	0	0
	ϕ_2	0	$1/\sqrt{2}$	$1/\sqrt{3}$	$1/\sqrt{3}$	0	0
	ϕ_3	0	0	$1/\sqrt{3}$	$1/\sqrt{3}$	$1/\sqrt{3}$	0
	ϕ_4	0	0	0	$1/\sqrt{3}$	$1/\sqrt{3}$	0
	ϕ_5	0	0	0	0	$1/\sqrt{3}$	0
Task 2	ϕ_1	1	$1/\sqrt{2}$	$1/\sqrt{3}$	$1/\sqrt{4}$	$1/\sqrt{5}$	0
	ϕ_2	0	$1/\sqrt{2}$	$1/\sqrt{3}$	$1/\sqrt{4}$	$1/\sqrt{5}$	0
	ϕ_3	0	0	$1/\sqrt{3}$	$1/\sqrt{4}$	$1/\sqrt{5}$	0
	ϕ_4	0	0	0	$1/\sqrt{4}$	$1/\sqrt{5}$	0
	ϕ_5	0	0	0	0	$1/\sqrt{5}$	0

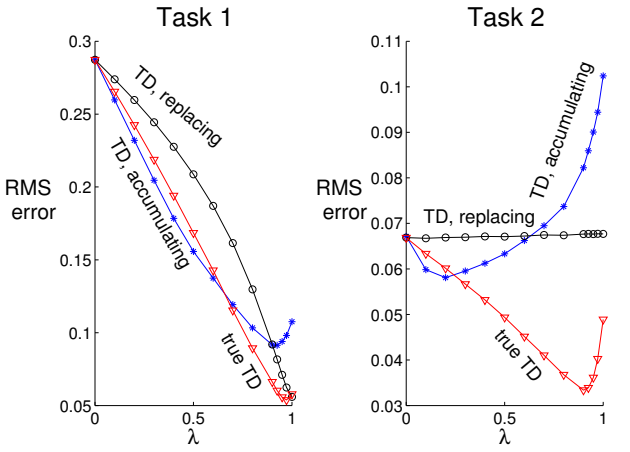

Figure 3. RMS error of state values at the end of each episode, averaged over the first 10 episodes, as well as 1000 independent runs, for different values of λ at the best value of α.

5. Control

The true online TD(λ) algorithm (Algorithm 2) can be easily modified for control. Simply using a feature vector consisting of state-action features (i.e., using $\boldsymbol{\phi}(s,a)$ instead of $\boldsymbol{\phi}(s)$), and changing the definition of δ_t to:

$$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}) - \gamma \hat{q}(S_t, A_t)$$

changes the algorithm to a true Sarsa(λ) algorithm.

Figure 4 compares true Sarsa(λ) with the other Sarsa(λ) implementations on the standard mountain car task (Sutton & Barto, 1998), using 10 tilings of each 10×10 tiles. Results are plotted for $\lambda = 0.9$ and a step-size

$$\alpha = \alpha_0 / 10$$

for α_0 from 0.2 to 2.0 with steps of 0.2. Clearing/no

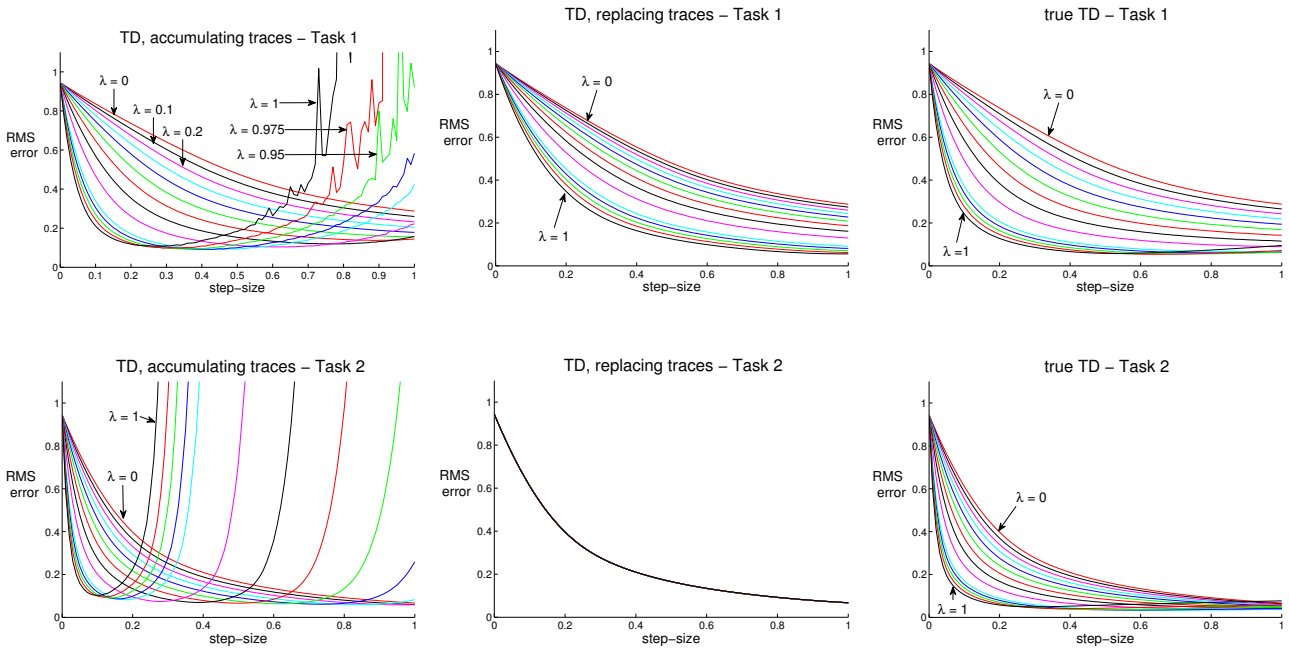

Figure 2. RMS error of state values at the end of each episode, averaged over the first 10 episodes, as well as 1000 independent runs, for different values of the step-size α and λ.

clearing refers to whether the trace values of non-selected actions are set to 0 (clearing) or not (no clearing), in case of replacing traces. The results demonstrate that the true online TD approach is also effective in a control setting.

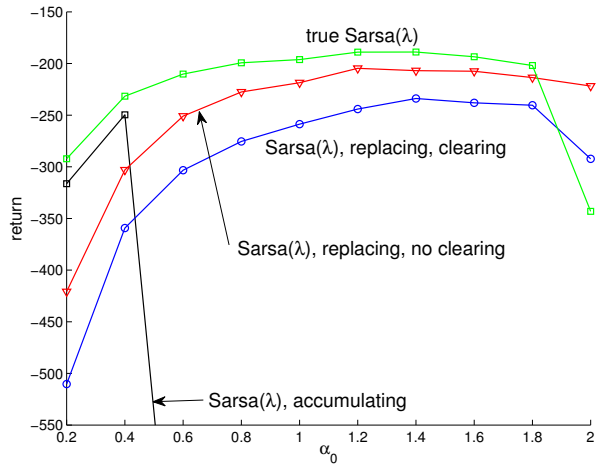

Figure 4. Average return over first 20 episodes on mountain car task for $\lambda = 0.9$ and different α_0. Results are averaged over 100 independent runs.

6. Conclusion

We solved an important open question in temporal-difference learning: whether it is possible to construct an online TD(λ) algorithm that is exactly equivalent to the forward view. We showed that not only is this possible, but an implementation can be made with a similar computational cost as the traditional TD(λ) implementation. Empirically, we demonstrated that this new implementation of TD(λ), which we call true online TD(λ), avoids divergence of value estimates that can occur under accumulating traces, while it does not suffer from the negative side-effects of replacing traces. In addition, we demonstrated that our approach is also effective in the control case. Based on the stability and consistent performance edge of true online TD(λ) over the traditional implementations, we expect that it will become the algorithm of choice for researchers relying on temporal-difference learning.

References

Kaelbling, L.P., Littman, M.L., and Moore, A.P. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

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

Singh, S.P., and Sutton, R.S. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1):123–158, 1996.

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

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