Off-policy TD(λ) with a true online equivalence

Hado van Hasselt **A. Rupam Mahmood** **Richard S. Sutton**
Reinforcement Learning and Artificial Intelligence Laboratory
University of Alberta, Edmonton, AB T6G 2E8 Canada

Abstract

Van Seijen and Sutton (2014) recently proposed a new version of the linear TD(λ) learning algorithm that is exactly equivalent to an online forward view and that empirically performed better than its classical counterpart in both prediction and control problems. However, their algorithm is restricted to on-policy learning. In the more general case of off-policy learning, in which the policy whose outcome is predicted and the policy used to generate data may be different, their algorithm cannot be applied. One reason for this is that the algorithm bootstraps and thus is subject to instability problems when function approximation is used. A second reason true online TD(λ) cannot be used for off-policy learning is that the off-policy case requires sophisticated importance sampling in its eligibility traces. To address these limitations, we generalize their equivalence result and use this generalization to construct the first online algorithm to be exactly equivalent to an off-policy forward view. We show this algorithm, named *true online GTD(λ)*, empirically outperforms GTD(λ) (Maei, 2011) which was derived from the same objective as our forward view but lacks the exact online equivalence. In the general theorem that allows us to derive this new algorithm, we encounter a new general eligibility-trace update.

1 Temporal difference learning

Eligibility traces improve learning in temporal-difference (TD) algorithms by efficiently propagating credit for later observations back to update earlier predictions (Sutton, 1988), and can help speed up learning significantly. A good way to interpret these traces, the extent of which is regulated by a trace parameter $\lambda \in [0, 1]$, is to consider the eventual updates to each prediction. For $\lambda = 1$ the update for the prediction at time t is similar to a Monte Carlo update towards the full return following t. For $\lambda = 0$ the prediction is updated toward only the immediately (reward) signal, and the rest of the return is estimated with the prediction at the next state. Such an interpretation is called a forward view, because it considers the effect of future observations on the updates. In practice, learning is often fastest for intermediate values of λ (Sutton & Barto, 1998).

Traditionally, the equivalence to a forward view was known to hold only when the predictions are updated offline. In practice TD algorithms are more commonly used online, during learning, but then this equivalence was only approximate. Recently, van Seijen and Sutton (2014) developed true online TD(λ), the first algorithm to be exactly equivalent to a forward view under online updating. For $\lambda = 1$ the updates by true online TD(λ) eventually become exactly equivalent to a Monte Carlo update towards the full return. As demonstrated by van Seijen and Sutton, such an online equivalence is more than a theoretical curiosity, and leads to lower prediction errors than when using the traditional TD(λ) algorithm that only achieves an offline equivalence. In this paper, we generalize this result and show exact online equivalences are possible for a wide range of forward views, leading to computationally efficient online algorithms by exploiting a new generic trace update.

A limitation of the true online TD(λ) algorithm by van Seijen and Sutton (2014) is that it is only applicable to on-policy learning, when the learned predictions correspond to the policy that is used to generate the data. Off-policy learning is important to be able to learn from demonstrations, to learn about many things at the same time (Sutton et al., 2011), and ultimately to learn about the unknown optimal policy. A natural next step is therefore to apply our general equivalence result to an off-policy forward view. We construct such a forward view and derive an equivalent new off-policy gradient TD algorithm, that we call *true online GTD(λ)*. This algorithm is constructed to be equivalent for $\lambda = 0$, by design, to the existing GTD(λ) algorithm (Maei, 2011). We demonstrate empirically that for higher λ the new algorithm is much better behaved due to its exact

equivalence to a desired forward view. In addition to the practical potential of the new algorithm, this demonstrates the usefulness of our general equivalence result and the resulting new trace update.

2 Problem setting

We consider a learning agent in an unknown environment where at each time step t the agent performs an action A_t after which the environment transitions from the current state S_t to the next state S_{t+1}. We do not assume the state itself can be observed and the agent instead observes a feature vector $\boldsymbol{\phi}_t \in \mathbb{R}^n$, which is typically a function of the state S_t such that $\boldsymbol{\phi}_t \doteq \boldsymbol{\phi}(S_t)$. The agent selects its actions according to a behavior policy b, such that $b(a|S_t)$ denotes the probability of selecting action $A_t = a$ in state S_t. Typically $b(a|s)$ depends on s through $\boldsymbol{\phi}(s)$.

After performing A_t, the agent observes a scalar (reward) signal R_{t+1} and the process can either terminate or continue. We allow for soft terminations, defined by a potentially time-varying state-dependent termination factor $\gamma_t \in [0, 1]$ (cf. Sutton, Mahmood, Precup & van Hasselt, 2014). With weight $1 - \gamma_{t+1}$ the process terminates at time $t + 1$ and R_{t+1} is considered the last reward in this episode. With weight γ_{t+1} we continue to the next state and observe $\boldsymbol{\phi}_{t+1} \doteq \boldsymbol{\phi}(S_{t+1})$. The agent then selects a new action A_{t+1} and this process repeats. A special case is the episodic setting where $\gamma_t = 1$ for all non-terminating times t and $\gamma_T = 0$ when the episode ends at time T. The termination factors are commonly called *discount factors*, because they discount the effect of later rewards.

The goal is to predict the sum of future rewards, discounted by the probabilities of termination, under a target policy π. The optimal prediction is thus defined for each state s by

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[\sum_{t=1}^{\infty} R_t \prod_{k=1}^{t} \gamma_k \mid S_0 = s\right] ,$$

where $\mathbb{E}_\pi[\,\cdot\,] \doteq \mathbb{E}[\,\cdot \mid A_t \sim \pi(\cdot|S_t), \forall t\,]$ is the expectancy conditional on the policy π. We estimate the values $v_\pi(s)$ with a parameterized function of the observed features. In particular we consider linear functions of the features, such that $\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \approx v_\pi(S_t)$ is the estimated value of the state at time t according to a weight vector $\boldsymbol{\theta}_t$. The goal is then to improve the predictions by updating $\boldsymbol{\theta}_t$.

We desire online algorithms with a constant $O(n)$ per-step complexity, where n is the number of features in $\boldsymbol{\phi}$. Such computational considerations are important in settings with a lot of data or when $\boldsymbol{\phi}_t$ is a large vector. For instance, we want our algorithms to be able to run on a robot with many sensors and limited on-board processing power.

3 General online equivalence between forward and backward views

We can think about what the ideal update would be for a prediction after observing all relevant future rewards and states. Such an update is called a forward view, because it depends on observations from the future. A concrete example is the on-policy Monte Carlo return, consisting of the discounted sum of all future rewards.

In practice, full Monte Carlo updates can have high variance. It can be better to augment the return with the then-current predictions at the visited states. When we continue after some time step t, with weight γ_{t+1}, we replace a portion of $1 - \lambda_{t+1}$ of the remaining return with our current prediction of this return at S_{t+1}. Making use of later predictions to update earlier predictions in this way is called *bootstrapping*. The process then continues to the next action and reward with total weight $\gamma_{t+1}\lambda_{t+1}$, where again we terminate with $1 - \gamma_{t+2}$ and then bootstrap with $1 - \lambda_{t+2}$, and so on. When $\lambda_{t+1} = 0$ we get the usual one-step TD return $R_{t+1} + \gamma_{t+1}\boldsymbol{\phi}_{t+1}^\top \boldsymbol{\theta}_t$. If $\lambda_t = 1$ for all t, we obtain a full (discounted) Monte Carlo return. In the on-policy setting, when we do not have to worry about deviations from the target policy, we can then update the prediction made at time t towards the on-policy λ-return defined by

$$G_t^\lambda = R_{t+1} + \gamma_{t+1}\left[(1 - \lambda_{t+1})\boldsymbol{\phi}_{t+1}^\top \boldsymbol{\theta}_t + \lambda_{t+1} G_{t+1}^\lambda\right] .$$

The discount factors γ_t are normally considered a property of the problem, but the bootstrap parameters λ_t can be considered tunable parameters. The full return (obtained for $\lambda = 1$) is an unbiased estimate for the value of the behavior policy, but its variance can be high. The value estimates are typically not unbiased, but can be considerably less variable. As such, one can interpret the λ parameters as trading off bias and variance. Typically, learning is fastest for intermediate values of λ.

If termination never occurs, G_t^λ is never fully defined. To construct a well-defined forward view, we can truncate the recursion at the current data horizon (van Seijen & Sutton, 2014; Sutton et al., 2014) to obtain interim λ-returns. If we have data up to time t, all returns are truncated as if $\lambda_t = 0$ and we bootstrap on the most recent value estimate $\boldsymbol{\phi}_t^\top \boldsymbol{\theta}_{t-1}$ of the current state. This gives us, for each $0 \leq k < t$

$$G_{k,t}^\lambda = R_{k+1} + \gamma_{k+1}\left[(1 - \lambda_{k+1})\boldsymbol{\phi}_{k+1}^\top \boldsymbol{\theta}_k + \lambda_{k+1} G_{k+1,t}^\lambda\right]$$

and $G_{t,t}^\lambda = \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_{t-1}$. In this definition of $G_{k,t}^\lambda$, for each time step j with $k < j \leq t$ the value of state S_j is estimated using $\boldsymbol{\phi}_j^\top \boldsymbol{\theta}_{j-1}$, because $\boldsymbol{\theta}_{j-1}$ is the most up-to-date weight vector at the moment we reach this state.

Using these interim returns, we can construct an interim forward view which, in contrast to conventional forward

views, can be computed before an episode has concluded or even if the episode never fully terminates. For instance, when we have data up to time t the following set of linear updates for all times $k < t$ is an interim forward view:

$$\boldsymbol{\theta}_{k+1}^t = \boldsymbol{\theta}_k^t + \alpha_k(G_{k,t}^\lambda - \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^t)\boldsymbol{\phi}_k\,, \;\; k < t\,, \qquad (1)$$

where $\boldsymbol{\theta}_0^t \doteq \boldsymbol{\theta}_0$ is the initial weight vector. The subscript on $\boldsymbol{\theta}_k^t$ (first index on $G_{k,t}^\lambda$) corresponds to the state for the kth update, the superscript (second index on $G_{k,t}^\lambda$) denotes the current data horizon.

The forward view (1) is well-defined and computable at every time t, but it is not very computationally efficient. For each new observation, when t increments to $t+1$, we potentially have to recompute all the updates, as $G_{k,t+1}^\lambda$ might differ from $G_{k,t}^\lambda$ for arbitrary many k. The resulting computational complexity is $O(nt)$ per time step, which is problematic when t becomes large. Therefore, forward views are not meant to be implemented as is. They serve as a conceptual update, in which we formulate what we want to achieve after observing the relevant data.

In the next theorem, we prove that for many forward views an efficient and fully equivalent *backward view* exists that exploits eligibility traces to construct online updates that use only $O(n)$ computation per time step, but that still result in exactly the same weight vectors. The theorem is constructive, allowing us to find such backward views automatically for a given forward view.

Theorem 1 (Equivalence between forward and backward views)**.** *Consider any forward view that updates towards some interim targets Y_k^t with*

$$\boldsymbol{\theta}_{k+1}^t = \boldsymbol{\theta}_k^t + \eta_k(Y_k^t - \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^t)\boldsymbol{\phi}_k + \boldsymbol{x}_k\,, \;\; 0 \le k < t\,,$$

where $\boldsymbol{\theta}_0^t = \boldsymbol{\theta}_0$ for some initial $\boldsymbol{\theta}_0$ and where $\boldsymbol{x}_k \in \mathbb{R}^n$ is any vector that does not depend on t. Assume that the temporal differences $Y_k^{t+1} - Y_k^t$ for different k are related through

$$Y_k^{t+1} - Y_k^t = c_k(Y_{k+1}^{t+1} - Y_{k+1}^t)\,, \;\; \forall k < t\,, \qquad (2)$$

where c_k is a scalar that does not depend on t. Then, the final weights $\boldsymbol{\theta}_t^t$ at each t are equal to the weights $\boldsymbol{\theta}_t$ as defined by $\boldsymbol{e}_0 = \eta_0\boldsymbol{\phi}_0$ and the backward view

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + (Y_t^{t+1} - Y_t^t)\boldsymbol{e}_t + \eta_t(Y_t^t - \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_t)\boldsymbol{\phi}_t + \boldsymbol{x}_t\,,\\
\boldsymbol{e}_t &= c_{t-1}\boldsymbol{e}_{t-1} + \eta_t(1 - c_{t-1}\boldsymbol{\phi}_t^\top \boldsymbol{e}_{t-1})\boldsymbol{\phi}_t\,, \;\; t > 0\,. \qquad (3)
\end{aligned}$$

Proof. We introduce the fading matrix $\boldsymbol{F}_t \doteq \boldsymbol{I} - \eta_t\boldsymbol{\phi}_t\boldsymbol{\phi}_t^\top$, such that $\boldsymbol{\theta}_{k+1}^t = \boldsymbol{F}_k\boldsymbol{\theta}_k^t + \eta_k Y_k^t\boldsymbol{\phi}_k$. We subtract $\boldsymbol{\theta}_t^t$ from $\boldsymbol{\theta}_{t+1}^{t+1}$ to find the change when t increments. Expanding $\boldsymbol{\theta}_{t+1}^{t+1}$, we get

$$\begin{aligned}
&\boldsymbol{\theta}_{t+1}^{t+1} - \boldsymbol{\theta}_t^t\\
&= \boldsymbol{F}_t\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t + \eta_t Y_t^{t+1}\boldsymbol{\phi}_t + \boldsymbol{x}_t\\
&= \boldsymbol{F}_t(\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t) + \eta_t Y_t^{t+1}\boldsymbol{\phi}_t + (\boldsymbol{F}_t - \boldsymbol{I})\boldsymbol{\theta}_t^t + \boldsymbol{x}_t\\
&= \boldsymbol{F}_t(\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t) + \eta_t Y_t^{t+1}\boldsymbol{\phi}_t - \eta_t\boldsymbol{\phi}_t\boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t^t + \boldsymbol{x}_t\\
&= \boldsymbol{F}_t(\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t) + \eta_t(Y_t^{t+1} - \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t^t)\boldsymbol{\phi}_t + \boldsymbol{x}_t\,. \qquad (4)
\end{aligned}$$

We now repeatedly expand both $\boldsymbol{\theta}_t^{t+1}$ and $\boldsymbol{\theta}_t^t$ to get

$$\begin{aligned}
&\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t\\
&= \boldsymbol{F}_{t-1}(\boldsymbol{\theta}_{t-1}^{t+1} - \boldsymbol{\theta}_{t-1}^t) + \eta_{t-1}(Y_{t-1}^{t+1} - Y_{t-1}^t)\boldsymbol{\phi}_{t-1}\\
&= \boldsymbol{F}_{t-1}\boldsymbol{F}_{t-2}(\boldsymbol{\theta}_{t-1}^{t+1} - \boldsymbol{\theta}_{t-1}^t)\\
&\quad + \eta_{t-2}(Y_{t-2}^{t+1} - Y_{t-2}^t)\boldsymbol{F}_{t-1}\boldsymbol{\phi}_{t-2}\\
&\quad + \eta_{t-1}(Y_{t-1}^{t+1} - Y_{t-1}^t)\boldsymbol{\phi}_{t-1}\\
&= \ldots \text{ (Expand until reaching } \boldsymbol{\theta}_0^{t+1} - \boldsymbol{\theta}_0^t = \boldsymbol{0}.)\\
&= \boldsymbol{F}_{t-1}\cdots\boldsymbol{F}_0(\boldsymbol{\theta}_0^{t+1} - \boldsymbol{\theta}_0^t)\\
&\quad + \sum_{k=0}^{t-1}\eta_k\boldsymbol{F}_{t-1}\cdots\boldsymbol{F}_{k+1}(Y_k^{t+1} - Y_k^t)\boldsymbol{\phi}_k\\
&= \sum_{k=0}^{t-1}\eta_k\boldsymbol{F}_{t-1}\cdots\boldsymbol{F}_{k+1}(Y_k^{t+1} - Y_k^t)\boldsymbol{\phi}_k\\
&= \sum_{k=0}^{t-1}\eta_k\boldsymbol{F}_{t-1}\cdots\boldsymbol{F}_{k+1}c_k(Y_{k+1}^{t+1} - Y_{k+1}^t)\boldsymbol{\phi}_k \;\; \text{(Using (2))}\\
&= \ldots \text{ (Apply (2) repeatedly.)}\\
&= c_{t-1}\underbrace{\sum_{k=0}^{t-1}\eta_k\left(\prod_{j=k}^{t-2}c_j\right)\boldsymbol{F}_{t-1}\cdots\boldsymbol{F}_{k+1}\boldsymbol{\phi}_k}_{\doteq\,\boldsymbol{e}_{t-1}}(Y_t^{t+1} - Y_t^t)\\
&= c_{t-1}\boldsymbol{e}_{t-1}(Y_t^{t+1} - Y_t^t)\,. \qquad (5)
\end{aligned}$$

The vector $\boldsymbol{e}_t$ can be computed with the recursion

$$\begin{aligned}
\boldsymbol{e}_t &= \sum_{k=0}^{t}\eta_k\left(\prod_{j=k}^{t-1}c_j\right)\boldsymbol{F}_t\cdots\boldsymbol{F}_{k+1}\boldsymbol{\phi}_k\\
&= \sum_{k=0}^{t-1}\eta_k\left(\prod_{j=k}^{t-1}c_j\right)\boldsymbol{F}_t\cdots\boldsymbol{F}_{k+1}\boldsymbol{\phi}_k + \eta_t\boldsymbol{\phi}_t\\
&= c_{t-1}\boldsymbol{F}_t\sum_{k=0}^{t-1}\eta_k\left(\prod_{j=k}^{t-2}c_j\right)\boldsymbol{F}_{t-1}\cdots\boldsymbol{F}_{k+1}\boldsymbol{\phi}_k + \eta_t\boldsymbol{\phi}_t\\
&= c_{t-1}\boldsymbol{F}_t\boldsymbol{e}_{t-1} + \eta_t\boldsymbol{\phi}_t\\
&= c_{t-1}\boldsymbol{e}_{t-1} + \eta_t(1 - c_{t-1}\boldsymbol{\phi}_t^\top\boldsymbol{e}_{t-1})\boldsymbol{\phi}_t\,.
\end{aligned}$$

We plug (5) back into (4) and obtain

$$\begin{aligned}
&\boldsymbol{\theta}_{t+1}^{t+1} - \boldsymbol{\theta}_t^t \\
&= c_{t-1}\boldsymbol{F}_t\boldsymbol{e}_{t-1}(Y_t^{t+1} - Y_t^t) + \eta_t(Y_t^{t+1} - \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t)\boldsymbol{\phi}_t + \boldsymbol{x}_t \\
&= (\boldsymbol{e}_t - \eta_t\boldsymbol{\phi}_t)(Y_t^{t+1} - Y_t^t) + \eta_t(Y_t^{t+1} - \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t)\boldsymbol{\phi}_t + \boldsymbol{x}_t \\
&= (Y_t^{t+1} - Y_t^t)\boldsymbol{e}_t + \eta_t(Y_t^t - \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t)\boldsymbol{\phi}_t + \boldsymbol{x}_t\,.
\end{aligned}$$

Because $\boldsymbol{\theta}_{0,t} \doteq \boldsymbol{\theta}_0$ for all t, the desired result follows through induction. □

The theorem shows that under condition (2) we can turn a general forward view into an equivalent online algorithm that only uses $O(n)$ computation per time step. Compared to previous work on forward/backward equivalences, this grants us two important things. First, the obtained equivalence is both online and exact; most previous equivalences were only exact under offline updating, when the weights are not updated during learning (Sutton & Barto, 1998; Sutton et al., 2014). Second, the theorem is constructive, and gives an equivalent backward view directly from a desired forward view, rather than having to prove such an equivalence in hindsight (as in, e.g., van Seijen & Sutton, 2014). This is perhaps the main benefit of the theorem: rather than relying on insight and intuition to construct efficient online algorithms, Theorem 1 can be used to derive an exact backward view directly from a desired forward view. We exploit this in Section 6 when we turn a desired off-policy forward view into an efficient new online off-policy algorithm.

We refer to traces of the general form (3) as *dutch traces*. The trace update can be interpreted as first shrinking the traces with c, for instance $c = \gamma\lambda$, and then updating the traces for the current state, $\boldsymbol{\phi}^\top\boldsymbol{e}$, towards one with a step size of η. In contrast, traditional accumulating traces, defined by $\boldsymbol{e}_t = c_{t-1}\boldsymbol{e}_{t-1} + \boldsymbol{\phi}_t$, add to the trace value of the current state rather than updating it toward one. This can cause the accumulating traces to grow large, potentially resulting in high-variance updates.

To demonstrate one advantage of Theorem 1, we apply it to the on-policy TD(λ) forward view defined by (1).

Theorem 2 (Equivalence for true online TD(λ))**.** *Define $\boldsymbol{\theta}_0^t = \boldsymbol{\theta}_0$. Then, $\boldsymbol{\theta}_t^t$ as defined by* (1) *equals $\boldsymbol{\theta}_t$ as defined by the backward view*

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

Proof. In Theorem 1, we substitute $\boldsymbol{x}_t = \boldsymbol{0}$, $c_t = \gamma\lambda$ and $Y_k^t = G_{k,t}^\lambda$, such that $Y_t^{t+1} - Y_t^t = \delta_t$ and $Y_t^t = \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_{t-1}$. The desired result follows immediately. □

The backward view in Theorem 2 is true online TD(λ), as proposed by van Seijen and Sutton (2014). Using Theorem 1, we have proved equivalence to its forward view with a few simple substitutions, whereas the original proof is much longer and more complex.

4 Off-policy learning

In this section, we turn to off-policy learning with function approximation. In constructing an off-policy forward view two issues arise that are not present in the on-policy setting. First, we need to estimate the value of a policy that is different than the one used to obtain the observations. Second, using a forward view such as (1) under off-policy sampling can cause it to be unstable, potentially resulting in divergence of the weights (Sutton et al., 2008). These issues can be avoided by constructing our off-policy algorithms to minimize a mean-squared projected Bellman error (MSPBE) with gradient descent (Sutton et al., 2009; Maei & Sutton, 2010; Maei, 2011).

The MSPBE was previously used to derive GTD(λ) (Maei, 2011), which is an online algorithm that can be used to learn off-policy predictions. GTD(λ) was not constructed to be exactly equivalent to any forward view and it is a natural question whether the algorithm can be improved from having such an equivalence, just as was the case with TD(λ) and true online TD(λ). In this section, we introduce an off-policy MSPBE and show how GTD(λ) can be derived. In the next section, we use the same MSPBE to construct a new off-policy forward view from which we will derive an exactly equivalent online backward view.

To obtain estimates for one distribution when the samples are generated under another distribution, we can weight the observations by the relative probabilities of these observations occurring under the target policy, as compared to the behavior distribution. This is called *importance sampling* (Rubinstein, 1981; Precup, Sutton & Singh, 2000). Recall that $b(a|s)$ and $\pi(a|s)$ denote the probabilities of selecting action a in state s according to the behavior policy and the target policy, respectively. After selecting an action A_t in a state S_t according to b, we observe a reward R_{t+1}. The expected value of this reward is $\mathbb{E}_b[R_{t+1}]$, but if we multiply the reward with the importance-sampling ratio $\rho_t \doteq \pi(A_t|S_t)/b(A_t|S_t)$ the expected value is

$$\begin{aligned}
\mathbb{E}_b[\rho_t R_{t+1}|S_t] &= \sum_a b(a|S_t)\frac{\pi(a|S_t)}{b(a|S_t)}\mathbb{E}[R_{t+1} \mid S_t, A_t = a] \\
&= \sum_a \pi(a|S_t)\mathbb{E}[R_{t+1} \mid S_t, A_t = a] \\
&= \mathbb{E}_\pi[R_{t+1} \mid S_t]\,.
\end{aligned}$$

Therefore $\rho_t R_{t+1}$ is an unbiased sample for the reward under the target policy. This technique can be applied to all the rewards and value estimates in a given λ-return.

For instance, if we want to obtain an unbiased sample for the reward under the target policy n steps after the current state S_t, the total weight applied to this reward should be $\rho_t \rho_{t+1} \cdots \rho_{t+n-1}$. An off-policy λ-return starting from state S_t is given by

$$G_t^{\lambda\rho}(\boldsymbol{\theta}) = \rho_t \Big(R_{t+1} + \gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}^\top \boldsymbol{\theta} + \gamma_{k+1}\lambda_{k+1} G_{k+1}^{\lambda}(\boldsymbol{\theta}) \Big) . \tag{6}$$

In contrast to G_t^λ, this return is defined as a function of a single weight vector $\boldsymbol{\theta}$. This is useful later, when we wish to determine the gradient of this return with respect to $\boldsymbol{\theta}$.

When using function approximation it is generally not possible to estimate the value of each state with full accuracy or, equivalently, to reduce the conditional expected TD error for each state to zero at the same time. More formally, let $v_{\boldsymbol{\theta}}$ be a parameterized value function defined by $v_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s)$ and let T_π^λ be a parametrized Bellman operator defined, for any $v : \{s\} \to \mathbb{R}$, by

$$(T_\pi^\lambda v)(s) = \mathbb{E}_\pi \big[R_1 + \gamma_1(1-\lambda_1) v(S_1) + \gamma_1 \lambda_1 (T_\pi^\lambda v)(S_1) \mid S_0 = s \big] .$$

In general, we then cannot achieve $v_{\boldsymbol{\theta}} = T_\pi^\lambda v_{\boldsymbol{\theta}}$, because $T_\pi^\lambda v_{\boldsymbol{\theta}}$ is not guaranteed to be a function that we can represent with our chosen function approximation. It is, however, possible to find the fixed point defined by

$$v_{\boldsymbol{\theta}} = \Pi T_\pi^\lambda v_{\boldsymbol{\theta}} . \tag{7}$$

where Πv is a projection of v into the space of representable functions $\{v_{\boldsymbol{\theta}} \mid \boldsymbol{\theta} \in \mathbb{R}^n\}$. Let d be the steady-state distribution of states under the behavior policy. The projection of any v is then defined by

$$\Pi v = v_{\boldsymbol{\theta}_v} , \qquad \text{where} \qquad \boldsymbol{\theta}_v = \arg\min_{\boldsymbol{\theta}} \| v_{\boldsymbol{\theta}} - v \|_d^2 ,$$

where $\| \cdot \|_d^2$ is a norm defined by $\|f\|_d^2 \doteq \sum_s d(s) f(s)^2$. Following Maei (2011), the projection is defined in terms of the steady-state distribution resulting from the behavior policy, which means that $d(s) = \lim_{t\to\infty} \mathbb{P}(S_t = s \mid A_j \sim b(\cdot|S_j), \forall j)$. This implies our objective weights the importance of the accuracy of the prediction in each state according to the relative frequency that this state occurs under the behavior policy, which is a natural choice for online learning.

The fixed point in (7) can be found by minimizing the MSPBE defined by (Maei, 2011)

$$\begin{aligned} J(\boldsymbol{\theta}) &= \| v_{\boldsymbol{\theta}} - \Pi T_\pi^\lambda v_{\boldsymbol{\theta}} \|_d^2 \\ &= \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta})\boldsymbol{\phi}_k]^\top \mathbb{E}_b\big[\boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top\big]^{-1} \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta})\boldsymbol{\phi}_k] , \end{aligned} \tag{8}$$

where $\delta_k^\pi(\boldsymbol{\theta}) \doteq (T_\pi^\lambda v_{\boldsymbol{\theta}})(S_k) - v_{\boldsymbol{\theta}}(S_k)$ and where the expectancies are with respect to the steady-state distribution d, as induced by the behavior policy b. The ideal gradient update for time step k is then

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{1}{2}\alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})|_{\boldsymbol{\theta}_k} , \tag{9}$$

where

$$\begin{aligned} &-\frac{1}{2}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})|_{\boldsymbol{\theta}_k} \\ &= -\mathbb{E}_b\big[\nabla_{\boldsymbol{\theta}} \delta_k^\pi(\boldsymbol{\theta}) \boldsymbol{\phi}_k^\top\big] \mathbb{E}_b\big[\boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top\big]^{-1} \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k)\boldsymbol{\phi}_k] \\ &= \mathbb{E}_b\Big[(\boldsymbol{\phi}_k - \nabla_{\boldsymbol{\theta}} G_k^{\lambda\rho}(\boldsymbol{\theta}))\boldsymbol{\phi}_k^\top\Big] \mathbb{E}_b\big[\boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top\big]^{-1} \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k)\boldsymbol{\phi}_k] \\ &= \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k)\boldsymbol{\phi}_k] \\ &\quad - \mathbb{E}_b\Big[\nabla_{\boldsymbol{\theta}} G_k^{\lambda\rho}(\boldsymbol{\theta})\boldsymbol{\phi}_k^\top\Big] \mathbb{E}_b\big[\boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top\big]^{-1} \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k)\boldsymbol{\phi}_k] \\ &= \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k)\boldsymbol{\phi}_k] - \mathbb{E}_b\Big[\nabla_{\boldsymbol{\theta}} G_k^{\lambda\rho}(\boldsymbol{\theta})\boldsymbol{\phi}_k\Big]^\top \boldsymbol{w}_* , \end{aligned} \tag{10}$$

with $G_k^{\lambda\rho}$ as defined in (6), and where

$$\boldsymbol{w}_* \doteq \mathbb{E}_b\big[\boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top\big]^{-1} \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k)\boldsymbol{\phi}_k] .$$

Update (9) can be interpreted as an expected forward view.

The derivation of the GTD(λ) algorithm proceeds by exploiting the expected equivalences (Maei, 2011)

$$\begin{aligned} &\mathbb{E}_b\big[\nabla_{\boldsymbol{\theta}} G_k(\boldsymbol{\theta})\boldsymbol{\phi}_k^\top\big] \\ &= \mathbb{E}_b\big[\rho_k\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}\boldsymbol{\phi}_k^\top\big] \\ &\quad + \mathbb{E}_b\big[\rho_k\gamma_{k+1}\lambda_{k+1}\nabla_{\boldsymbol{\theta}} G_{k+1}(\boldsymbol{\theta})\boldsymbol{\phi}_k^\top\big] \\ &= \mathbb{E}_b\big[\rho_k\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}\boldsymbol{\phi}_k^\top\big] \\ &\quad + \mathbb{E}_b\big[\rho_{k-1}\gamma_k\lambda_k\nabla_{\boldsymbol{\theta}} G_k(\boldsymbol{\theta})\boldsymbol{\phi}_{k-1}^\top\big] \\ &= \mathbb{E}_b\big[\rho_k\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}\boldsymbol{\phi}_k^\top\big] \\ &\quad + \mathbb{E}_b\big[\rho_{k-1}\gamma_k\lambda_k\rho_k\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}\boldsymbol{\phi}_{k-1}^\top\big] \\ &\quad + \mathbb{E}_b\big[\rho_{k-1}\gamma_k\lambda_k\rho_k\gamma_{k+1}\lambda_{k+1}\nabla_{\boldsymbol{\theta}} G_{k+1}(\boldsymbol{\theta})\boldsymbol{\phi}_{k-1}^\top\big] \\ &= \mathbb{E}_b\big[\rho_k\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}\boldsymbol{\phi}_k^\top\big] \\ &\quad + \mathbb{E}_b\big[\rho_{k-1}\gamma_k\lambda_k\rho_k\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}\boldsymbol{\phi}_{k-1}^\top\big] \\ &\quad + \mathbb{E}_b\big[\rho_{k-2}\gamma_{k-1}\lambda_{k-1}\rho_{k-1}\gamma_k\lambda_k\nabla_{\boldsymbol{\theta}} G_k(\boldsymbol{\theta})\boldsymbol{\phi}_{k-2}^\top\big] \\ &= \ldots \text{(Repeat until we reach } \boldsymbol{\phi}_0.) \\ &= \mathbb{E}_b\Bigg[\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1} \underbrace{\rho_k \sum_{j=0}^{k} \Bigg(\prod_{i=j+1}^{k} \rho_{i-1}\gamma_i\lambda_i \Bigg) \boldsymbol{\phi}_j^\top}_{\doteq (\boldsymbol{e}_k^\nabla)^\top}\Bigg] \\ &= \mathbb{E}_b\big[\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}(\boldsymbol{e}_k^\nabla)^\top\big] , \end{aligned} \tag{11}$$

and, similarly, $\mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k)\boldsymbol{\phi}_k] = \mathbb{E}_b[\delta_k(\boldsymbol{\theta}_k)\boldsymbol{e}_k^\nabla]$, where

$$\begin{aligned} \boldsymbol{e}_t^\nabla &= \rho_t(\gamma_t\lambda_t\boldsymbol{e}_{t-1}^\nabla + \boldsymbol{\phi}_t) , \\ \delta_t(\boldsymbol{\theta}) &= R_{t+1} + \gamma_{t+1}\boldsymbol{\phi}_{t+1}^\top\boldsymbol{\theta} - \boldsymbol{\phi}_t^\top\boldsymbol{\theta} . \end{aligned} \tag{12}$$

The auxiliary vector $\boldsymbol{w}_t \approx \boldsymbol{w}_*$ can be updated with least mean squares (LMS) (Sutton et al., 2009; Maei, 2011), using the sample $\delta_t(\boldsymbol{\theta}_t)\boldsymbol{e}_t^\nabla \approx \mathbb{E}_b[\delta_t(\boldsymbol{\theta}_t)\boldsymbol{e}_t^\nabla] = \mathbb{E}_b[\delta_t^\pi(\boldsymbol{\theta}_t)\boldsymbol{\phi}_t]$

and the update

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \beta_t \delta_t(\boldsymbol{\theta}_t)\boldsymbol{e}_t^{\nabla} - \beta_t \boldsymbol{\phi}_t^\top \boldsymbol{w}_t \boldsymbol{\phi}_t \,.$$

The complete GTD(λ) algorithm is then defined by[1]

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\boldsymbol{\phi}_{t+1}^\top \boldsymbol{\theta}_t - \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_t \,, \\
\boldsymbol{e}_t^{\nabla} &= \rho_t(\gamma_t \lambda_t \boldsymbol{e}_{t-1}^{\nabla} + \boldsymbol{\phi}_t) \,, \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t \delta_t \boldsymbol{e}_t^{\nabla} - \alpha_t \gamma_{t+1}(1-\lambda_{t+1}) \boldsymbol{w}_t^\top \boldsymbol{e}_t^{\nabla} \boldsymbol{\phi}_{t+1} \,, \\
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \beta_t \delta_t \boldsymbol{e}_t^{\nabla} - \beta_t \boldsymbol{\phi}_t^\top \boldsymbol{w}_t \boldsymbol{\phi}_t \,.
\end{aligned}$$

5 An off-policy forward view

In this section, we define an off-policy forward view which we turn into a fully equivalent backward view in the next section, using Theorem 1. GTD(λ) is derived by first turning an expected forward view into an expected backward view, and then sampling. We propose instead to sample the expected forward view directly and then invert the sampled forward view into an equivalent online backward view. This way we obtain an exact equivalence between forward and backward views instead of the expected equivalence of GTD(λ). This was previously not known to be possible, but it has the advantage that we can use the precise (potentially discounted and bootstrapped) sample returns consisting of all future rewards and state values in each update. This can result in more accurate predictions, as confirmed by our experiments in Section 7.

The new forward view derives from the MSPBE, as defined in (8), and more specifically from the gradient update defined by (9) and (10). To find an implementable interim forward view, we need sampled estimates of all three parts in (10). We discuss each of these parts separately.

Our interim forward view is defined in terms of a data horizon t, so the gradient of the MSPBE is taken to $\boldsymbol{\theta}_k^t$ rather than $\boldsymbol{\theta}_k$. Furthermore, δ_k^π is defined as the error between a λ-return and a current estimate, and therefore we need to construct an interim λ-return. To estimate the first term of (10) we therefore need an estimate for $\mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k^t)\boldsymbol{\phi}_k] = \mathbb{E}_b[G_{k,t}^{\lambda\rho} - \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^t]$, for some suitably defined $G_{k,t}^{\lambda\rho}$.

The variance of off-policy updates is often lower when we weight the errors (that is, the difference between the return and the current estimate) with the importance-sampling ratios, rather than weighting the returns (Sutton et al., 2014). Let $\delta_k = R_{k+1} + \gamma_{k+1}\boldsymbol{\phi}_{k+1}^\top \boldsymbol{\theta}_k - \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1}$ denote a one-step TD error. The on-policy return used in the forward view (1) can then be written as a sum of such errors:

$$G_{k,t}^{\lambda} = \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1} + \sum_{j=k}^{t-1} \left(\prod_{i=k+1}^{j} \gamma_i \lambda_i \right) \delta_j \,.$$

[1]Dann, Neumann and Peters (2014) call this algorithm TDC(λ), but we use the original name by Maei (2011).

We apply the importance-sampling weights to the one-step TD errors, rather than just to the reward and bootstrapped value estimate.[2] This does not affect the expected value, because $\mathbb{E}_b\big[\rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1} \mid S_k\big] = \mathbb{E}_b\big[\boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1} \mid S_k\big]$, but it can have a beneficial effect on the variance of the resulting updates. A sampled off-policy error is then

$$G_{k,t}^{\lambda\rho} - \rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^t \approx \mathbb{E}_b\big[\delta_k^\pi(\boldsymbol{\theta}_k^t)\boldsymbol{\phi}_k\big] \,, \tag{13}$$

where

$$G_{k,t}^{\lambda\rho} \doteq \rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1} + \rho_k \sum_{j=k}^{t-1} \left(\prod_{i=k+1}^{j} \gamma_i \lambda_i \rho_i \right) \delta_j \,.$$

An equivalent recursive definition for $G_{k,t}^{\lambda\rho}$ is

$$\begin{aligned}
G_{k,t}^{\lambda\rho} = \rho_k \Big(& R_{k+1} + \gamma_{k+1}(1 - \lambda_{k+1}\rho_{k+1}) \boldsymbol{\phi}_{k+1}^\top \boldsymbol{\theta}_k \\
& + \gamma_{k+1}\lambda_{k+1} G_{k+1,t}^{\lambda\rho} \Big) \,,
\end{aligned} \tag{14}$$

for $k < t$, and $G_{t,t}^{\lambda\rho} \doteq \rho_t \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_{t-1}$. In the on-policy case, when $\rho_k = 1$ for all k, $G_{k,t}^{\lambda\rho}$ reduces exactly to $G_{k,t}^{\lambda}$, as used in the forward view (1) for true online TD(λ). Furthermore, $\mathbb{E}_b[G_{k,t}^{\lambda\rho} \mid S_k = s] = \mathbb{E}_\pi[G_{k,t}^{\lambda} \mid S_k = s]$ for any s.

For the second term in (10), which can be thought of as the gradient correction term, we need an estimate $\boldsymbol{w}_k \approx \boldsymbol{w}_*$. As in the derivation of GTD(λ), we use a LMS update. Assuming we have data up to t, the ideal forward-view update for $\boldsymbol{w}_k^t$ is then

$$\boldsymbol{w}_{k+1}^t = \boldsymbol{w}_k^t + \beta_k(\delta_{k,t}^{\lambda\rho} - \boldsymbol{\phi}_k^\top \boldsymbol{w}_k^t)\boldsymbol{\phi}_k \,, \tag{15}$$

for some appropriate sample $\delta_{k,t}^{\lambda\rho} \approx \mathbb{E}_b[\delta_k^\pi(\boldsymbol{\theta}_k)]$. A natural interim estimate is defined by

$$\delta_{k,t}^{\lambda\rho} = \rho_k(\delta_k + \gamma\lambda\delta_{k+1,t}^{\lambda\rho}) \,, \tag{16}$$

where $\delta_{t,t}^{\lambda\rho} = 0$ and

$$\delta_k = R_{k+1} + \gamma \boldsymbol{\theta}_k^\top \boldsymbol{\phi}_{k+1} - \boldsymbol{\theta}_k^\top \boldsymbol{\phi}_k \,.$$

This is not the only possible way to estimate $\boldsymbol{w}_*$, but this choice ensures the resulting algorithm is equivalent to GTD(0) when $\lambda = 0$, allowing us to investigate the effects of the true online equivalence and the resulting new trace updates in some isolation without having to worry about other potential differences between the algorithms. In the next section we construct an equivalent backward view for (15) to compute the sequence $\{\boldsymbol{w}_t\}$, where $\boldsymbol{w}_t = \boldsymbol{w}_t^t$, $\forall t$.

[2]For the PTD(λ) and PQ(λ) algorithms, Sutton et al. (2014) propose another weighting based on weighting flat return errors containing multiple rewards. In contrast, our weighting is chosen to be consistent with GTD(λ). True online versions of PTD(λ) and PQ(λ) exist, but we do not consider them further in this paper.

Finally, we use the expected equivalence proved in (11), and then sample to obtain

$$\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}(\boldsymbol{e}_k^{\nabla})^\top \approx \mathbb{E}_b\big[\nabla_{\boldsymbol{\theta}} G_k(\boldsymbol{\theta})\boldsymbol{\phi}_k^\top\big]\ , \quad (17)$$

with $\boldsymbol{e}_k^{\nabla}$ as defined in (12).

We now have all the pieces to state the off-policy forward view for $\boldsymbol{\theta}$. We approximate the expected forward view as defined by (9) and (10) by using the sampled estimates (13), (17) and $\boldsymbol{w}_k = \boldsymbol{w}_k^k \approx \boldsymbol{w}_*$, with $\boldsymbol{w}_k^k$ as defined by (15). This gives us the interim forward view

$$\begin{aligned}\boldsymbol{\theta}_{k+1}^t = \boldsymbol{\theta}_k^t &+ \alpha_k(G_{k,t}^{\lambda\rho} - \rho_k\boldsymbol{\phi}_k^\top\boldsymbol{\theta}_k^t)\boldsymbol{\phi}_k \quad (18)\\ &- \alpha_k\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}\boldsymbol{w}_k^\top\boldsymbol{e}_k^{\nabla}\ ,\end{aligned}$$

with $G_{k,t}^{\lambda\rho}$ as defined in (14).

6 Backward view: true online GTD(λ)

In this section, we apply Theorem 1 to convert the off-policy forward view as given by (18) into an efficient online backward view. First, we consider $\boldsymbol{w}$.

Theorem 3 (Auxiliary vectors)**.** *The vector $\boldsymbol{w}_t^t$, as defined by the forward view in* (15)*, is equal to $\boldsymbol{w}_t$ as defined by the backward view*

$$\begin{aligned}\boldsymbol{e}_t^{\boldsymbol{w}} &= \rho_{t-1}\gamma_t\lambda_t\boldsymbol{e}_{t-1}^{\boldsymbol{w}} + \beta_t(1-\rho_{t-1}\gamma_t\lambda_t\boldsymbol{\phi}_t^\top\boldsymbol{e}_{t-1}^{\boldsymbol{w}})\boldsymbol{\phi}_t\ ,\\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \rho_t\delta_t\boldsymbol{e}_t^{\boldsymbol{w}} - \beta_t\boldsymbol{\phi}_t^\top\boldsymbol{w}_t\boldsymbol{\phi}_t\ ,\end{aligned}$$

where $\boldsymbol{e}_0^{\boldsymbol{w}} = \beta_0\boldsymbol{\phi}_0$, $\boldsymbol{w}_0 = \boldsymbol{w}_0^t, \forall t$, and

$$\delta_t \doteq R_{t+1} + \gamma\boldsymbol{\phi}_{t+1}^\top\boldsymbol{\theta}_t - \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t\ .$$

Proof. We apply Theorem 1 by substituting $\boldsymbol{\theta}_t = \boldsymbol{w}_t$, $\eta_t = \beta_t$, $\boldsymbol{x}_t = \boldsymbol{0}$ and $Y_k^t = \delta_{k,t}^{\lambda\rho}$, as defined in (16). Then

$$\delta_{k,t+1}^{\lambda\rho} - \delta_{k,t}^{\lambda\rho} = \rho_k\gamma_{k+1}\lambda_{k+1}(\delta_{k+1,t+1}^{\lambda\rho} - \delta_{k+1,t}^{\lambda\rho})\ ,$$

which implies $c_k = \rho_k\gamma_{k+1}\lambda_{k+1}$. Finally, $Y_t^t = \delta_{t,t}^{\lambda\rho} = 0$ and $Y_t^{t+1} - Y_t^t = \delta_{t,t+1}^{\lambda\rho} = \rho_t\delta_t$. Inserting these substitutions into the backward view in Theorem 1 immediately yields the backward view in the current theorem. □

Theorem 4 (True online GTD(λ))**.** *For any t, the weight vector $\boldsymbol{\theta}_t^t$ as defined by forward view in* (18) *is equal to $\boldsymbol{\theta}_t$, as defined by the backward view*

$$\begin{aligned}\boldsymbol{e}_t &= \rho_t(\gamma_t\lambda_t\boldsymbol{e}_{t-1} + \alpha_t(1-\rho_t\gamma_t\lambda_t\boldsymbol{\phi}_t^\top\boldsymbol{e}_{t-1})\boldsymbol{\phi}_t)\ ,\\ \boldsymbol{e}_t^{\nabla} &= \rho_t(\gamma_t\lambda_t\boldsymbol{e}_{t-1}^{\nabla} + \boldsymbol{\phi}_t)\ ,\\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \delta_t\boldsymbol{e}_t + (\boldsymbol{e}_t - \alpha_t\rho_t\boldsymbol{\phi}_t)(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})^\top\boldsymbol{\phi}_t\\ &\qquad - \alpha_t\gamma_{t+1}(1-\lambda_{t+1})\boldsymbol{w}_t^\top\boldsymbol{e}_t^{\nabla}\boldsymbol{\phi}_{t+1}\ ,\end{aligned}$$

with $\boldsymbol{w}_t$ and δ_t as defined in Theorem 3.

Proof. Again, we apply Theorem 1. Substitute $\eta_t = \rho_t\alpha_t$, $\boldsymbol{x}_k = -\alpha_k\gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}_{k+1}\boldsymbol{w}_k^\top\boldsymbol{e}_k^{\nabla}$, $Y_t^t = \boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t$ and

$$Y_k^t = R_{k+1} + \gamma_{k+1}(1-\lambda_{k+1}\rho_{k+1})\boldsymbol{\theta}_k^\top\boldsymbol{\phi}_{k+1} + \gamma\lambda G_{k+1,t}^{\lambda\rho}\ .$$

This last substitution implies

$$Y_k^{t+1} - Y_k^t = \gamma_{k+1}\lambda_{k+1}\rho_{k+1}(Y_{k+1}^{t+1} - Y_{k+1}^t)\ ,$$

so that $c_k = \gamma_{k+1}\lambda_{k+1}\rho_{k+1}$. Furthermore,

$$\begin{aligned}Y_t^{t+1} - Y_t^t &= R_{t+1} + \gamma\boldsymbol{\theta}_t^\top\boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t\\ &= \delta_t + (\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})^\top\boldsymbol{\phi}_t\ .\end{aligned}$$

Applying Theorem 1 with these substitutions, and replacing $\boldsymbol{w}_t^t$ with the equivalent $\boldsymbol{w}_t$, yields the backward view

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + (\delta_t + (\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})^\top\boldsymbol{\phi}_t)\boldsymbol{e}_t\\ &\quad + \alpha_t\rho_t(\boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t - \boldsymbol{\theta}_t^\top\boldsymbol{\phi}_t)\boldsymbol{\phi}_t\\ &\quad - \alpha_t\gamma_{t+1}(1-\lambda_{t+1})\boldsymbol{w}_t^\top\boldsymbol{e}_t^{\nabla}\boldsymbol{\phi}_{t+1}\ ,\\ &= \boldsymbol{\theta}_t + \delta_t\boldsymbol{e}_t + (\boldsymbol{e}_t - \alpha_t\rho_t\boldsymbol{\phi}_t)\boldsymbol{\phi}_t^\top(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})\\ &\quad - \alpha_t\gamma_{t+1}(1-\lambda_{t+1})\boldsymbol{w}_t^\top\boldsymbol{e}_t^{\nabla}\boldsymbol{\phi}_{t+1}\ ,\end{aligned}$$

where $\boldsymbol{e}_0 = \alpha_0\rho_0\boldsymbol{\phi}_0$ and

$$\boldsymbol{e}_t = \rho_t\gamma_t\lambda_t\boldsymbol{e}_{t-1} + \alpha_t\rho_t(1-\rho_t\gamma_t\lambda_t\boldsymbol{e}_{t-1}^\top\boldsymbol{\phi}_t)\boldsymbol{\phi}_t\ .\ \square$$

True online GTD(λ) algorithm is then defined by

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma\boldsymbol{\phi}_{t+1}^\top\boldsymbol{\theta}_t - \boldsymbol{\phi}_t^\top\boldsymbol{\theta}_t\ ,\\ \boldsymbol{e}_t &= \rho_t(\gamma_t\lambda_t\boldsymbol{e}_{t-1} + \alpha_t(1-\rho_t\gamma_t\lambda_t\boldsymbol{\phi}_t^\top\boldsymbol{e}_{t-1})\boldsymbol{\phi}_t)\ ,\\ \boldsymbol{e}_t^{\nabla} &= \rho_t(\gamma_t\lambda_t\boldsymbol{e}_{t-1}^{\nabla} + \boldsymbol{\phi}_t)\ ,\\ \boldsymbol{e}_t^{\boldsymbol{w}} &= \rho_{t-1}\gamma_t\lambda_t\boldsymbol{e}_{t-1}^{\boldsymbol{w}} + \beta_t(1-\rho_{t-1}\gamma_t\lambda_t\boldsymbol{\phi}_t^\top\boldsymbol{e}_{t-1}^{\boldsymbol{w}})\boldsymbol{\phi}_t\ ,\\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \delta_t\boldsymbol{e}_t + (\boldsymbol{e}_t - \alpha_t\rho_t\boldsymbol{\phi}_t)(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})^\top\boldsymbol{\phi}_t\\ &\qquad - \alpha_t\gamma_{t+1}(1-\lambda_{t+1})\boldsymbol{w}_t^\top\boldsymbol{e}_t^{\nabla}\boldsymbol{\phi}_{t+1}\ ,\\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \rho_t\delta_t\boldsymbol{e}_t^{\boldsymbol{w}} - \beta_t\boldsymbol{\phi}_t^\top\boldsymbol{w}_t\boldsymbol{\phi}_t\ .\end{aligned}$$

The traces $\boldsymbol{e}_t$ and $\boldsymbol{e}_t^{\boldsymbol{w}}$ are dutch traces. The trace $\boldsymbol{e}_t^{\nabla}$ is an accumulating trace that follows from the gradient correction, as discussed in Section 4. It might be possible to adapt the forward view to replace $\boldsymbol{e}_t^{\nabla}$ with $\boldsymbol{e}_t$. This is already possible in practice and in preliminary experiments the resulting algorithm performed similar to true online GTD(λ). A more detailed investigation of this possibility is left for future work.

For $\lambda = 0$ the algorithm reduces to

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t\rho_t\delta_t\boldsymbol{\phi}_t - \alpha_t\rho_t\gamma_{t+1}\boldsymbol{w}_t^\top\boldsymbol{\phi}_t\boldsymbol{\phi}_{t+1}\ ,\\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \beta_t\rho_t\delta_t\boldsymbol{\phi}_t - \beta_t\boldsymbol{\phi}_t^\top\boldsymbol{w}_t\boldsymbol{\phi}_t\ ,\end{aligned}$$

which is precisely GTD(0).[3]

[3] The on-policy variant of this algorithm, with $\rho_t = 1$ for all t, is known as TDC (Sutton et al., 2009; Maei, 2011).

7 Experiments

We compare true online GTD(λ) to GTD(λ) empirically in various settings. The main goal of the experiments is to test the intuition that true online GTD(λ) should be more robust to high step sizes and high λ, due to its true online equivalence and better behaved traces. This was shown to be the case for true online TD(λ) (van Seijen & Sutton, 2014), and the experiments serve to verify that this extends to the off-policy setting with true online GTD(λ). This is relevant because it implies true online GTD(λ) should then be easier to tune in practice, and because these parameters can effect the limiting performance of the algorithms as well.

Both algorithms optimize the MSPBE, as given in (8), which is a function of λ. When the state representation is of poor quality, the solution that minimizes the MSPBE can still have a high mean-sqared error (MSE): $\|v_{\boldsymbol{\theta}} - v^\pi\|_d^2$. This means that with a low λ we are not always guaranteed to reach a low MSE, even asymptotically. The closer λ is to one, the closer the MSPBE becomes to the MSE, with equality for $\lambda = 1$. In practice this implies that sometimes we need a high λ to be able to obtain an sufficiently accurate predictions, even if we run the algorithms a long time.

To illustrate these points, we investigate a fairly simple problem. The problem setting is a random walk consisting of 15 states that can be thought to lie on a horizontal line. In each state we have two actions: move one state to the left, or one state to the right. If we move left in the left-most state, s_1, we bounce back into that state. If we move right in the right-most state, s_{15}, the episode ends and we get a reward of $+1$. On all other time steps, the reward is zero. Each episode starts in s_1, which is the left-most state.

This problem setting is similar to the one used by van Seijen and Sutton (2014), with three differences. First, we use 15 rather than 11 states, but this makes little difference to the conclusions. Second, we turn it into an off-policy learning problem, as we describe in a moment. Third, we use different state representations. This last point is because we want to test the performance of the algorithm not just with features that can accurately represent the value function, as used by van Seijen and Sutton, but also with features that cannot reduce the MSE all the way to zero.

In the original problem, there was a 0.9 probability of moving right in each state (van Seijen & Sutton, 2014). Here, we interpret these probabilities as begin due to a behavior policy that selects the 'right' action with probability 0.9. Then, we formulate a target policy that want to move right more often, with probability 0.95. The stochastic target policy demonstrates that our algorithm is applicable to arbitrary off-policy learning tasks, and that the results do not depend on the target policy being deterministic. We did also test the performance for a deterministic policy that moves right always and the results are similar to those given

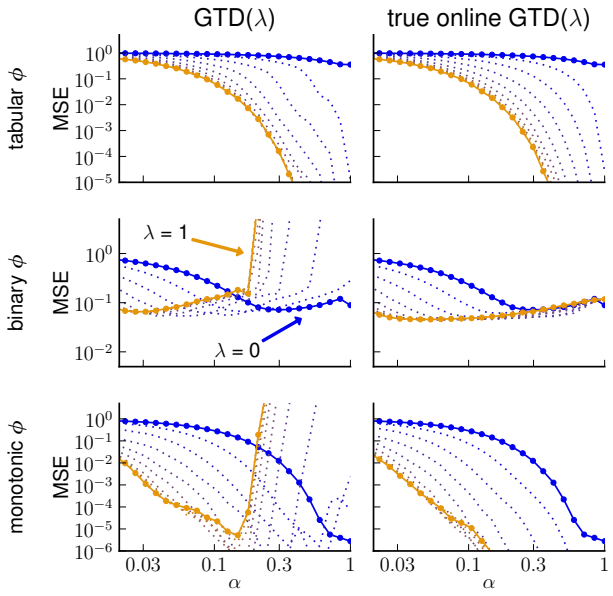

Figure 1: The MSE on the random walk of GTD(λ) (left column) and true online GTD(λ) (right column). The x-axis shows α, and the different lines are for different λ, with $\lambda = 0$ in blue and $\lambda = 1$ in orange. The top row is for 15 tabular features, the middle row for 4 binary features, and the bottom row for 2 monotonic features. The MSE is minimized over β.

below. Because this is an episodic task, $\gamma = 1$.

As stated above, we define three different state representations. In the first task, we use tabular features, such that $\boldsymbol{\phi}(s_i)$ is a vector of 15 elements, with the ith element equal to one and all other elements equal to zero. In the second task the state number is turned into a binary representation, such that $\boldsymbol{\phi}(s_1) = (0,0,0,1)^\top$, $\boldsymbol{\phi}(s_2) = (0,0,1,0)^\top$, $\boldsymbol{\phi}(s_3) = (0,0,1,1)^\top$, and so on up to $\boldsymbol{\phi}(s_{15}) = (1,1,1,1)^\top$. The features are then normalized to be unit vectors, such that for instance $\boldsymbol{\phi}(s_3) = (0,0,\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})^\top$ and $\boldsymbol{\phi}(s_{15}) = (\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2})^\top$. In our final representation, we use one monotonically increasing feature and one monotonically decreasing feature, such that $\boldsymbol{\phi}(s_i) = (\frac{14-i+1}{14},\frac{i-1}{14})^\top$ for all i. These features were not normalized.

For α the range of parameters was from 2^{-8} to 1 with steps in the exponent of 0.25 so that $\alpha \in \{2^{-8}, 2^{-7.75}, \ldots, 1\}$. The secondary step size β was varied over the same range, with the addition of $\beta = 0$. The trace parameter λ was varied from 0 to $1 - 2^{-10} \approx 0.999$ with steps of -1 in the exponent and with the addition of $\lambda = 1$, such that $\lambda \in \{0, 1-2^{-1}, \ldots, 1-2^{-9}, 1-2^{-10}, 1\}$.

The MSE (averaged over 20 repetitions) after 10 episodes for all three representations are shown in Figure 1. The left graphs all correspond to GTD(λ) and the plots on the right

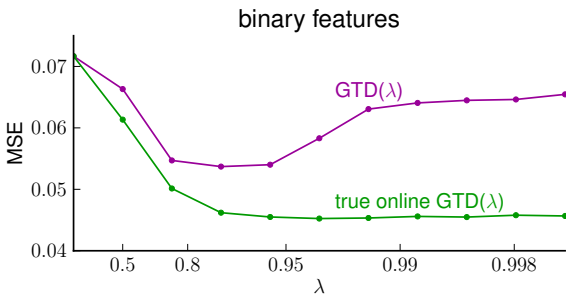

Figure 2: The MSE on the random walk for different λ of GTD(λ) and true online GTD(λ) for optimized α and β and binary features.

are for true online GTD(λ). Each graph show the MSE as a function of α, with different lines for different values of λ of which the extremes are highlighted ($\lambda = 0$ is blue; $\lambda = 1$ is orange). In all cases, the MSE was minimized for β, but this secondary step size had little impact on the performance at all in these problems. Note that the blue lines in the pair of graphs in each row are exactly equal, because by design the algorithms are equivalent for $\lambda = 0$.

In the top plots, the tabular representation was used and we see that especially with high λ both algorithms reach low prediction errors. This demonstrates that indeed learning can be faster with higher λ. When using function approximation, in the middle and bottom graphs, the benefit of having an online equivalence to a well-defined forward view becomes apparent. For both representations, the performance of GTD(λ) with higher λ begins to deteriorate around $\alpha = 0.2$. In contrast, true online GTD(λ) performs well even for $\alpha = \lambda = 1$. Note the log scale of the y-axis; the difference in MSE is many orders of magnitude.

In practice it is not always possible to fully tune the algorithmic parameters and therefore the robustness of true online GTD(λ) to different settings is important. However, it is still interesting to see what the best performance could be for a fully tuned algorithm. Therefore, in Figure 2 we show the MSE as a function of λ when minimized over both α and β. For all λ, true online GTD(λ) outperforms GTD(λ).

8 Discussion

The main theoretical contribution of this paper is a general theorem for equivalences between forward and backward views. The theorem allows us to find an efficient fully equivalent online algorithm for a desired forward view. The theorem is as general as required and as specific as possible for all applications of it in this paper, and in its current form it is limited to forward views for which an $O(n)$ backward view exists. The theorem can be generalized further, to include recursive (off-policy) LSTD(λ) (Boyan, 1999) and other algorithms that can be formulated in terms of forward views (cf. Geist & Scherrer, 2014; Dann, Neumann & Peters, 2014), but we did not investigate these extensions.

We used Theorem 1 to construct a new off-policy algorithm named true online GTD(λ), which is the first TD algorithm to have an exact online equivalence to a off-policy forward view. We constructed this forward view to maintain equivalence to the existing GTD(λ) algorithm for $\lambda = 0$. The forward view we proposed is not the only possible, and in particular it will be interesting to investigate different methods of importance sampling. We could for instance use the importance sampling as proposed by Sutton et al. (2014). We did construct the resulting online algorithm, and in preliminary tests its performance was similar to true online GTD(λ). Likewise, if desired, it is possible to obtain a full online equivalence to off-policy Monte Carlo for $\lambda = 1$ by constructing a forward view that achieves this. For instance we could use a similar forward view as used in the paper, but then apply the importance-sampling ratios only to the returns rather than to the errors. For now, it remains an open question what the best off-policy forward view is.

True online GTD(λ) is limited to state-value estimates. It is straightforward to construct a corresponding algorithm for action values, similar to the correspondence between GTD(λ) and GQ(λ) (Maei & Sutton, 2010; Maei, 2011) and between PTD(λ) and PQ(λ) (Sutton et al., 2014). We leave such an extension for future work.

Acknowledgments

The authors thank Joseph Modayil, Harm van Seijen and Adam White for fruitful discussions that helped improve the quality of this work. This work was supported by grants from Alberta Innovates – Technology Futures, the National Science and Engineering Research Council of Canada, and the Alberta Innovates Centre for Machine Learning.

References

Boyan, J. A., (1999). Least-squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning*, pp. 49–56.

Dann, C., Neumann, G., & Peters, J. (2014). Policy evaluation with temporal differences: A survey and comparison. In *Journal of Machine Learning Research 15*:809–883.

Geist, M., & Scherrer, B. (2014). Off-policy learning with eligibility traces: A survey. In *Journal of Machine Learning Research 15*:289–333.

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

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

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

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo method*. New York, Wiley.

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

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

Sutton, R. S., Mahmood, A. R. , Precup, D., & 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*. JMLR W&CP 32(2).

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

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., & 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*, pp. 761–768.

Sutton, R. S., Szepesvári, Cs., & Maei, H. R. (2008). A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. In *Advances in Neural Information Processing Systems 21*, pp. 1609–1616. MIT Press.

van Seijen, H., & Sutton, R. S. (2014). True online TD(λ). In *Proceedings of the 31st International Conference on Machine Learning*. JMLR W&CP 32(1):692–700.