A new Q(λ) with interim forward view and Monte Carlo equivalence

Richard S. Sutton, A. Rupam Mahmood {SUTTON,ASHIQUE}@CS.UALBERTA.CA
Reinforcement Learning and Artificial Intelligence Laboratory, University of Alberta, Edmonton, AB T6G 2E8 Canada

Doina Precup DPRECUP@CS.MCGILL.CA
School of Computer Science, McGill University, Montréal, QC H3A 0G4 Canada

Hado van Hasselt VANHASSE@CS.UALBERTA.CA
Reinforcement Learning and Artificial Intelligence Laboratory, University of Alberta, Edmonton, AB T6G 2E8 Canada

Abstract

Q-learning, the most popular of reinforcement learning algorithms, has always included an extension to eligibility traces to enable more rapid learning and improved asymptotic performance on non-Markov problems. The λ parameter smoothly shifts on-policy algorithms such as TD(λ) and Sarsa(λ) from a pure bootstrapping form ($\lambda=0$) to a pure Monte Carlo form ($\lambda=1$). In off-policy algorithms, including Q(λ), GQ(λ), and off-policy LSTD(λ), the λ parameter is intended to play the same role, but does not; on every exploratory action these algorithms bootstrap regardless of the value of λ, and as a result they fail to approximate Monte Carlo learning when $\lambda = 1$. It may seem that this is inevitable for any online off-policy algorithm; if updates are made on each step on which the target policy is followed, then how could just the right updates be 'un-made' upon deviation from the target policy? In this paper, we introduce a new version of Q(λ) that does exactly that, without significantly increased algorithmic complexity. En route to our new Q(λ), we introduce a new derivation technique based on the forward-view/backward-view analysis familiar from TD(λ) but extended to apply at every time step rather than only at the end of episodes. We apply this technique to derive first a new off-policy version of TD(λ), called PTD(λ), and then our new Q(λ), called PQ(λ).

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

1. Off-policy eligibility traces

Eligibility traces (Sutton 1988, Singh & Sutton 1996) are the mechanism by which temporal-difference (TD) algorithms such as Q-learning (Watkins 1989), Sarsa(λ) (Rummery 1995), and TD(λ) (Sutton 1988) escape the tyranny of the time step. In their simplest, one-step forms, these algorithms pass credit for an error back only to the single state preceding the error. If the time step is short, then the backward propagation of accurate values can be quite slow. With traces, credit is passed back to multiple preceding states, and learning is often significantly faster (Singh & Dayan 1998).

In on-policy TD algorithms with traces, such as TD(λ) and Sarsa(λ), the assignment of credit to previous states fades for more distantly preceding states according to the bootstrapping parameter $\lambda \in [0, 1]$. If $\lambda = 0$, then the traces fall to zero immediately and the one-step form of the algorithm is produced. If $\lambda = 1$, then the traces fade maximally slowly, no bootstrapping is done, and the resulting algorithm can be considered a *Monte Carlo* algorithm because in the limit it is almost equivalent to updating towards the complete observed return. The equivalence to Monte Carlo is exact in the episodic case under off-line updating (Sutton & Barto 1998) and for the new "true-online TD(λ)" algorithm recently introduced by van Seijen and Sutton (2014), and it approaches exactness in the conventional online case as the step-size parameter approaches zero.

Many *off-policy* TD algorithms also use traces, but less successfully. For example, Watkins's (1989) Q(λ) assigns credit to preceding state–action pairs, fading with temporal distance, but only as long as greedy actions are taken. On the first non-greedy (exploratory) action, the traces are cut and the algorithm bootstraps (updates its estimates as a function of other estimates) regardless of the value of λ. This is a problem because we would like to use $\lambda = 1$ to specify non-bootstrapping, Monte Carlo updates.

Newer off-policy algorithms, including GQ(λ) (Maei & Sutton 2010), GTD(λ) (Maei 2011), off-policy LSTD(λ) (Yu 2010), and others (Geist & Scherrer 2014) extend better to function approximation, but have essentially the same weakness: when the action taken deviates from the target policy all these algorithms cut off their traces and bootstrap. Peng's (1993) Q(λ) has a different problem. It never cuts the traces, but fails to converge to the optimal value function: for $\lambda = 1$ it approximates *on-policy* Monte Carlo. None of the existing off-policy algorithms make Monte Carlo updates when $\lambda = 1$; they do not update toward complete returns even in the most favorable case of episodic, off-line updating and infinitesimal step size. More generally, their degree of bootstrapping cannot be set via λ independently of the actions taken.

It might seem impossible for an online off-policy TD algorithm to do anything other than bootstrap when the action taken deviates from the target policy. By the time of deviation many online updates have already been made, but the deviation means that they cannot be counted as due to the target policy. The right thing to do, if $\lambda = 1$, is to somehow undo the earlier updates, but this has always seemed impossible to do online without greatly increasing the computational complexity of the algorithm. In this paper we show that, surprisingly, it can be done with very little additional memory and computation. In the case of linear function approximation, one additional vector is needed to hold *provisional weights*, which record the portion of the main weights that may need to be removed later upon deviation, or otherwise adjusted based on the degree of match to the target policy. With this technique we produce the first off-policy TD learning algorithms that are also Monte Carlo algorithms at $\lambda = 1$. We call the new algorithms PTD(λ) and PQ(λ) (where the 'P' alludes to their use of the distinctive mechanism of Provisional weights).

The main contributions of this paper are 1) the two new off-policy algorithms PTD(λ) and PQ(λ); 2) a new forward view of these algorithms that ensures Monte Carlo equivalence at $\lambda = 1$; 3) the notion of an "interim" forward view and a technique for using it to derive and prove equivalence of backward-view algorithms; and 4) applications of the technique to derive and prove equivalences for PTD(λ) and PQ(λ). A smaller contribution is that the new forward view allows arbitrary λ and γ that are general functions of state. The new algorithms are described in terms of linear function approximation but, like the original Q-learning, they are guaranteed convergent only for the tabular case; the extension to gradient-TD methods (a la Maei 2011) seems straightforward but is left to future work (see van Hasselt, Mahmood & Sutton, 2014). The flow of this paper is to first present the new forward view, then the derivation technique, and finally the new algorithms.

2. A new forward view

A key part of previous analyses of eligibility traces has been the notion of a *forward view* specifying the ideal update in terms of future events (Sutton & Barto 1998, Chapter 7). It can then sometimes be shown that a causal algorithm (a *backward view*) is mathematically equivalent in its overall effect to the forward view. In this section we incrementally introduce notation and ideas leading to a new forward-view algorithm expressed as a backup diagram and an error equation. By design, the new forward view becomes a Monte Carlo algorithm when $\lambda = 1$, and thus so will any equivalent backward-view algorithm. We will show in subsequent sections that PTD(λ) is such an equivalent backward-view algorithm and that PQ(λ) is equivalent to an analogous forward view for state–action values. The new forward view is significantly more general than previous forward views in that 1) it applies to the off-policy case as well as to the on-policy case; 2) it applies at each time step rather than just at the end of an episode (i.e., it is an *interim* forward view); and 3) it includes general functional forms for termination and bootstrapping.

We consider a continuing setting in which an agent and environment interact at each of a series of time steps, $t = 0, 1, 2, \ldots$. At each step t, the environment is in state S_t and generates a feature vector $\boldsymbol{\phi}(S_t) \in \mathbb{R}^n$. The agent then chooses an action A_t from a distribution defined by its fixed *behavior policy* $b(\cdot|S_t)$ (the only influence of S_t on b is typically via $\boldsymbol{\phi}(S_t)$, but we do not require this). The environment then emits a reward $R_{t+1} \in \mathbb{R}$ and transitions to a new state S_{t+1}, and the process continues. The next state and expected reward are assumed to be chosen from a joint distribution that depends only on the preceding state and action (the Markov assumption). We consider the problem of finding a weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$ such that

$$\boldsymbol{\theta}^\top \boldsymbol{\phi}(s) \approx \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} R_{t+1} \prod_{k=1}^{t} \gamma(S_k) \middle| S_0 = s \right] \qquad (1)$$

in some sense (for some norm), where the expectation is conditional on actions being selected according to an alternative policy π, called the *target policy*, and $\gamma(\cdot)$ is an almost arbitrary function from the state space to $[0, 1]$. We will refer to $\gamma(\cdot)$ as the *termination function* because when it falls to zero it terminates the quantity whose expectation is being estimated (as in Sutton 1995, Modayil et al. 2014, Sutton et al. 2011, Maei 2011). The only restriction we place on the termination function (and on the environment) is that $\prod_{k=1}^{\infty} \gamma(S_{t+k}) = 0$ w.p.1, $\forall t$.

If the two policies are the same, $\pi = b$, then the setting is called *on-policy*. If $\pi \neq b$, then we have the *off-policy* case, which is harder. For now we assume the two policies do not change over time (ultimately we will want to include control algorithms, in which at least the target policy changes).

We also assume that the behavior policy overlaps the target policy in the sense that $\pi(a|s) > 0 \implies b(a|s) > 0, \forall a,s$. The degree of deviation of the target policy from the behavior policy at each time t is captured by the importance-sampling ratio (Precup et al. 2000, 2001):

$$\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}. \tag{2}$$

If the behavior policy is not known, (for example, if it's a human expert), then $b(A_t|S_t)$ is taken to be 1.

In the off-policy case it is awkward to seek the approximation (1) from data because the expectation is under π whereas the data is due to b. Some actions chosen by b will match π, but not an infinite number in succession. This is where it is useful to interpret $\gamma(\cdot)$ as termination rather than as discounting. That is, we consider any trajectory, $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$ as having terminated partly after one step, at S_1, with degree of termination $1-\gamma(S_1)$, yielding a return of R_1, and partly after two steps, at S_2, with degree of termination $\gamma(S_1)(1-\gamma(S_2))$, yielding a return of $R_1 + R_2$. The third partial termination would have a degree of termination of $\gamma_1\gamma_2(1-\gamma_3)$ (where here we have switched to the shorthand $\gamma_t = \gamma(S_t)$) and would yield a return of $R_1 + R_2 + R_3$. Notice how in this partial termination view we end up with undiscounted, or *flat*, returns with no factors of $\gamma(\cdot)$. More importantly, we get short returns that have actually terminated, to various degrees, after each finite number of steps, rather than having to wait for an infinite number of steps to obtain a return as suggested by the discounting perspective.

Now consider the off-policy aspect and the role of the importance-sampling ratios. The first return, consisting of just R_1, needs to be weighted not only by its degree of termination, $1-\gamma_1$, but also by the importance sampling ratio ρ_0. This ratio will emphasize or de-emphasize this return depending on whether the action A_0 was more or less common under π than it is under b. If it was an action that would rarely be done under the target policy π and yet is in fact common (under the behavior policy b) then this instance of it will be de-emphasized proportionally to make up for its prevalence. If it would be common under the target policy but is rare (under the behavior policy) then its emphasis is pumped up proportionally to balance its infrequency. The overall weight on this return should then be the product of the degree of termination and the importance sampling ratio, or $\rho_0(1-\gamma_1)$. The weighting of the second return, $R_1 + R_2$, will depend on its degree of termination, $\gamma_1(1-\gamma_2)$ times the product of two importance sampling ratios, $\rho_0\rho_1$. The first ratio takes into account the match of the target and behavior policies for A_0, and the second takes into account the match for A_1. The overall weighting on the second term is $\rho_0\rho_1\,\gamma_1(1-\gamma_2)$. Continuing similarly, the weighting on the third flat return, $R_1 + R_2 + R_3$, is $\rho_0\rho_1\rho_2\,\gamma_1\gamma_2(1-\gamma_3)$.

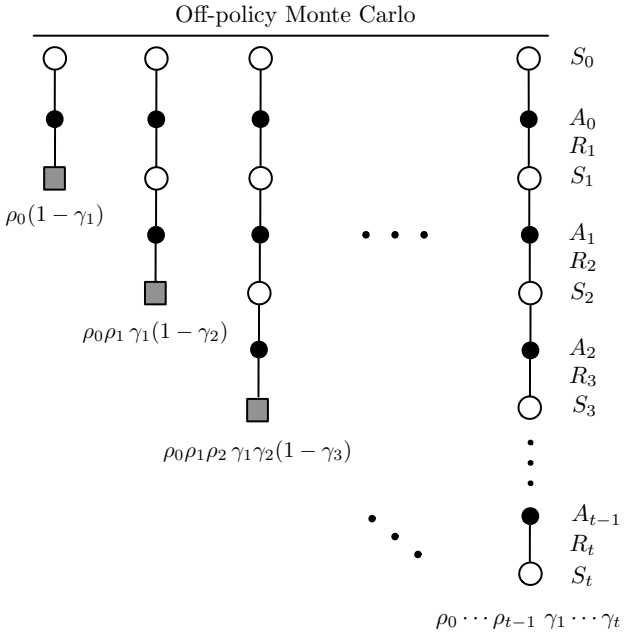

Figure 1. The backup diagram and interim forward view of an off-policy Monte Carlo algorithm (ρ but no λ).

These weights and the backup diagrams for these three returns are shown in the first three columns of Figure 1. Such figures are the standard way of diagramming a "complex" backup—a backup composed of of multiple sub-backups, one per column, each done in parallel with the weighting written below them (e.g., see Sutton & Barto 1998, Chapter 7). An important difference, however, is that here the sub-backups all use flat returns, whereas previous backup diagrams always implicitly included discounting.

The last column of the diagram is another small innovation. This sub-backup is an *interim forward view*, meaning that it looks ahead not to the next episode boundary, but to an arbitrary finite horizon time t. The horizon could coincide with the episode boundary (or be arbitrarily far out in the future), so interim forward views are a strict generalization of conventional forward views. Normally we will chose the horizon t to be something like the current limit of available data—the current time. Interim forward views are used to talk about the updates that could have been done in the past using all the data up to the current time. Note that the last sub-backup ends in a non-terminal state. According to the convention of backup diagrams, this means that bootstrapping using the current approximation is done at time t.

Next we introduce full bootstrapping into our new forward view. We generalize the conventional scalar λ parameter to an arbitrary bootstrapping function $\lambda(\cdot)$ from the state space to $[0,1]$ (as in Sutton & Singh 1994, Maei & Sutton 2010, cf. Kearns & Singh 2000, Downey & Sanner 2010).

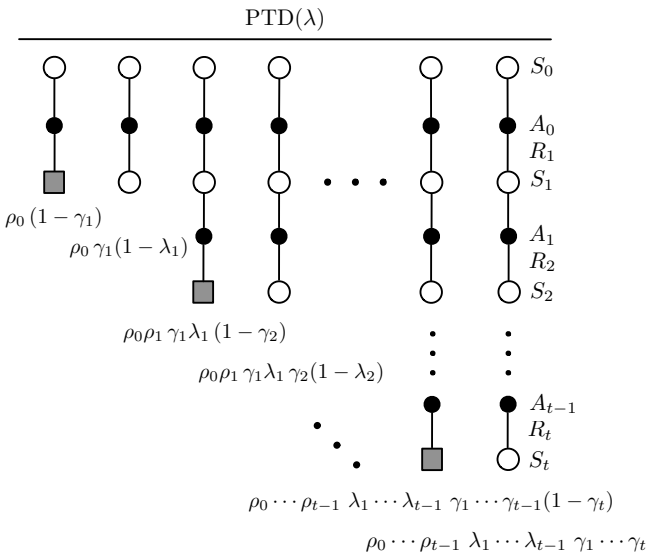

Figure 2. The backup diagram corresponding to our new forward view and to the PTD(λ) algorithm. Note that if $\lambda_t = 1\;\forall t$, then all the weightings are identical to those in Figure 1.

The bootstrapping factor at time t, $\lambda_t = \lambda(S_t)$, represents the degree of bootstrapping upon arrival in S_t, just as the termination factor $\gamma_t = \gamma(S_t)$ represents the degree of termination upon arriving there. The new backup diagram is shown in Figure 2. Each terminating sub-backup of the Monte Carlo backup in Figure 1 has been split into a terminating sub-backup and a bootstrapping sub-backup. Each sub-backup is a way in which the return can finish and no longer be subject to reweighting based on subsequent importance sampling ratios. The overall backup is best read from left to right. In the first two columns, the return can terminate in S_1 with a weighting of $\rho_0(1-\gamma_1)$, yielding a return of just R_1 or, to the extent that that does not occur, it can bootstrap in S_1 yielding a return of $R_1 + \boldsymbol{\theta}^\top\boldsymbol{\phi}(S_1)$. Bootstrapping can only be done to the extent that there was no prior deviation, ρ_0, or termination, γ_1, and only to the extent that λ_1 is less than 1; the overall weighting is thus $\rho_0\gamma_1(1-\lambda_1)$. To the extent that neither of these occur we go on similarly to the two-step sub-backups, which yield returns of $R_1 + R_2$ or $R_1 + R_2 + \boldsymbol{\theta}^\top\boldsymbol{\phi}(S_2)$ respectively if they apply, or we go on to the three-step sub-backups, and so on until we reach the last bootstrapping sub-backup at the horizon, at which bootstrapping is complete as if $\lambda_t = 0$, yielding a return of $R_1 + \cdots + R_t + \boldsymbol{\theta}^\top\boldsymbol{\phi}(S_t)$ with weighting $\rho_0\cdots\rho_{t-1}\,\lambda_1\cdots\lambda_{t-1}\,\gamma_1\cdots\gamma_t$. Notice that, if $\lambda_t = 1\;\forall t$, then the weightings on all the bootstrapping sub-backups, save the last, are zero, and all the other weightings are identical to those in Figure 1. This proves that this forward view is identical in this special case to off-policy Monte Carlo.

There is one final important issue before we can reduce these ideas and backups to equations. The preceding discussion has suggested that there is a total amount of weight that must be split up among the component backups of the complex backup. This is not totally incorrect. The expectation of the total weight along a trajectory is always one by construction, but the total weight for a sample trajectory will often be significantly larger or smaller than one. This is entirely caused by the importance sampling ratios. Previous complex backups, including off-policy ones like Q-learning, have not had to deal with this issue. We deal with it here by switching from viewing the backup diagram as a way of generating a *return* to a way of generating an *error*. We compute the weighting and then use it to weight the complete error rather than just the return portion of the error. This change does not affect the expected update of the algorithm, but we believe that it significantly reduces the average variance (Precup et al. 2001).

As an example of how error weighting can reduce variance, consider a state in which two actions are available, after each of which a single reward is received and the episode terminates. If the first action is taken, then the reward is $+1$, whereas, if the second action is taken, then the reward is -1. Suppose further that the target policy is to always select the first action, whereas the behavior policy is to select from the two actions randomly with equal probability. Then, on half of the episodes the first action will be taken and we will have a weighting of $\rho = 2$ and a return of $G = +1$, while on the other half of the episodes the second action will be taken and we will have a weighting of $\rho = 0$ and a return of $G = -1$. Now suppose that our current prediction is already correct at $\boldsymbol{\theta}^\top\boldsymbol{\phi} = 1$, and consider the updates that result under return weighting and under error weighting. If the weighting is applied to the return, then the update is according to $\rho G - \boldsymbol{\theta}^\top\boldsymbol{\phi}$, which is either $+1$ or -1 with equal probability. Whereas, if the weighting is applied to the error, then the update is according to $\rho(G - \boldsymbol{\theta}^\top\boldsymbol{\phi})$, which is always zero. Although both schemes have the same expected value, the error weighting clearly has lower variance in this example.

In our complex backup (Figure 2) there are two kinds of sub-backups. The first, ending with a terminal state, corresponds to a multi-step flat error without bootstrapping:

$$\epsilon_k^t = R_{k+1} + R_{k+2} + \cdots + R_t - \boldsymbol{\theta}^\top\boldsymbol{\phi}(S_k). \qquad (3)$$

The second sub-backup of each pair, ending with a non-terminal state, corresponds to a multi-step flat TD error, that is, a multi-step flat error with bootstrapping:

$$\bar{\delta}_k^t = R_{k+1} + \cdots + R_t + \boldsymbol{\theta}^\top\boldsymbol{\phi}(S_t) - \boldsymbol{\theta}^\top\boldsymbol{\phi}(S_k). \qquad (4)$$

We denote the overall error corresponding to the forward view in Figure 2 by $\delta_{0,t}^{\lambda\rho}$ (the error (δ) including both bootstrapping (λ) and importance-sampling weightings (ρ) at

time 0 with data to horizon t). It can be written compactly as a nested sum of pairs of terms, each corresponding to a sub-backup of the figure:

$$\begin{aligned}\delta_{0,t}^{\lambda\rho} &= \rho_0\Big[(1-\gamma_1)\epsilon_0^1 + \gamma_1(1-\lambda_1)\bar\delta_0^1 \\ &\quad + \gamma_1\lambda_1\rho_1\Big[(1-\gamma_2)\epsilon_0^2 + \gamma_2(1-\lambda_2)\bar\delta_0^2 \\ &\quad + \gamma_2\lambda_2\rho_2\Big[(1-\gamma_3)\epsilon_0^3 + \gamma_3(1-\lambda_3)\bar\delta_0^3 \\ &\quad + \cdots\gamma_{t-1}\lambda_{t-1}\rho_{t-1}\big[(1-\gamma_t)\epsilon_0^t + \gamma_t\bar\delta_0^t\big]\Big]\Big]\Big] \\ &= \rho_0\sum_{i=1}^{t-1} C_0^{i-1}\Big[(1-\gamma_i)\epsilon_0^i + \gamma_i(1-\lambda_i)\bar\delta_0^i\Big] \\ &\quad + \rho_0 C_0^{t-1}\left[(1-\gamma_t)\epsilon_0^t + \gamma_t\bar\delta_0^t\right],\end{aligned}$$

where

$$C_k^t = \prod_{i=k+1}^{t} \gamma_i\lambda_i\rho_i\,. \tag{5}$$

This completes our specification of the forward view error from time step 0. If we carefully repeat the same steps starting from an arbitrary time step $k < t$, then we get the following general form for the new forward-view error:

$$\begin{aligned}\delta_{k,t}^{\lambda\rho} &= \rho_k\sum_{i=k+1}^{t-1} C_k^{i-1}\Big[(1-\gamma_i)\epsilon_k^i + \gamma_i(1-\lambda_i)\bar\delta_k^i\Big] \\ &\quad + \rho_k C_k^{t-1}\left[(1-\gamma_t)\epsilon_k^t + \gamma_t\bar\delta_k^t\right].\end{aligned} \tag{6}$$

Finally, the error $\delta_{k,t}^{\lambda\rho}$ is used to form the forward-view increment in the weight vector at step k with horizon t:

$$\Delta\boldsymbol{\theta}_{k,t}^F = \alpha\delta_{k,t}^{\lambda\rho}\boldsymbol{\phi}(S_k), \tag{7}$$

where $\alpha > 0$ is a step-size parameter, and $\Delta\boldsymbol{\theta}_{k,t}^F$ denotes the update that would ideally be made at step k given data only up to horizon t. Of course, once the data stream has advanced to $t+1$, a different update would have been ideal at step k. This is one reason we do not actually make these updates online. Another is that we will be seeking to establish equivalences, as in previous work, for *off-line* updating, in which, as here, we compute forward-view updates, but do not actually change the weight vector as we go along. The exact form of the equivalences, and our technique for deriving backward-view updates that could be done online, is presented in the next section.

3. Interim equivalence technique

In this section we introduce a new technique for moving from an arbitrary interim forward view to an equivalent backward view, that is, to a causal mechanistic algorithm. The technique is general, but the resultant algorithm may or may not have an efficient implementation using incrementally computed auxiliary variables such as eligibility traces. In the next two sections we use this technique to derive efficient implementations of our new algorithms PTD(λ) and PQ(λ). A precursor to our general technique was previously developed by van Seijen & Sutton (2014).

In demonstrations of equivalence between forward and backward views, it is conventional to assume *off-line updating* (Sutton & Barto 1998), in which updates to the weight vector are computed on each time step but collected on the side and not used to change the weight vector until the end of the episode. Equivalence means that, by the end of the episode, the sum of the updates of the forward view is equal to the sum of the updates of the backward view. That is, there is a forward view update $\Delta\boldsymbol{\theta}_k^F$ and a backward view update $\Delta\boldsymbol{\theta}_k^B$ for each step k of the episode, which may depend on the weight vector $\boldsymbol{\theta}$, assumed constant during the episode. Equivalence then means that $\sum_{k=0}^{T-1}\Delta\boldsymbol{\theta}_k^F = \sum_{k=0}^{T-1}\Delta\boldsymbol{\theta}_k^B$ where the episode is taken to start at step 0 and terminate at step T.

An interim equivalence is slightly stronger. We still assume off-line updating, but now we require an equivalence not just at the end of the episode but at every possible horizon t within the episode. That is, we seek

$$\sum_{k=0}^{t-1}\Delta\boldsymbol{\theta}_k^B = \sum_{k=0}^{t-1}\Delta\boldsymbol{\theta}_{k,t}^F, \qquad \forall t, \tag{8}$$

where $\Delta\boldsymbol{\theta}_{k,t}^F$ denotes a forward view update for step k with horizon t. This is a tall order, and it is not always possible to achieve it with a computationally efficient algorithm. Nevertheless, the following theorem shows that it can always be achieved and suggests how the backward-view update might be derived from the forward-view.

Theorem 1 (Interim equivalence technique)**.** *The algorithm with backward-view update*

$$\Delta\boldsymbol{\theta}_t^B = \sum_{k=0}^{t}\Delta\boldsymbol{\theta}_{k,t+1}^F - \sum_{k=0}^{t-1}\Delta\boldsymbol{\theta}_{k,t}^F. \tag{9}$$

is equivalent in the interim sense (8) *with respect to the forward-view update* $\Delta\boldsymbol{\theta}_{k,t}^F$.

Proof. Define $\boldsymbol{\theta}_t^F = \sum_{k=0}^{t-1}\Delta\boldsymbol{\theta}_{k,t}^F$. Then $\sum_{k=0}^{t-1}\Delta\boldsymbol{\theta}_k^B = \sum_{k=0}^{t-1}(\boldsymbol{\theta}_{k+1}^F - \boldsymbol{\theta}_k^F) = \boldsymbol{\theta}_t^F - \boldsymbol{\theta}_0^F = \boldsymbol{\theta}_t^F = \sum_{k=0}^{t-1}\Delta\boldsymbol{\theta}_{k,t}^F$. □

4. PTD(λ) and TD(λ)

We now apply the interim equivalence technique to derive PTD(λ) from the new forward view developed in Section 2. Continuing from (9), we have

$$\Delta\boldsymbol{\theta}_t^B = \Delta\boldsymbol{\theta}_{t,t+1}^F + \sum_{k=0}^{t-1}\Big[\Delta\boldsymbol{\theta}_{k,t+1}^F - \Delta\boldsymbol{\theta}_{k,t}^F\Big]$$

$$= \alpha\delta^{\lambda\rho}_{t,t+1}\boldsymbol{\phi}_t + \alpha\sum_{k=0}^{t-1}\left[\delta^{\lambda\rho}_{k,t+1} - \delta^{\lambda\rho}_{k,t}\right]\boldsymbol{\phi}_k, \quad (10)$$

by (7), where here, and in much of the following, we use the notational shorthand $\boldsymbol{\phi}_t = \boldsymbol{\phi}(S_t)$. The term in brackets above suggests that it would be useful to have a form for the forward view error in terms of itself with a one-step longer horizon. Note that increasing the horizon by one just adds two more sub-backups on the right in Figure 2, with a small adjustment to the weighting on the original final backup. This suggests that the recursive form might be reasonably simple, and in fact it is. In the supplementary material we derive that, for $k < t$,

$$\delta^{\lambda\rho}_{k,t+1} = \delta^{\lambda\rho}_{k,t} + \rho_k C^t_k \delta_t + (\rho_t - 1)\gamma_t\lambda_t\rho_k C^{t-1}_k \bar{\delta}^t_k, \quad (11)$$

where δ_t is the classical TD error:

$$\delta_t = R_{t+1} + \gamma_{t+1}\boldsymbol{\theta}^\top\boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}^\top\boldsymbol{\phi}_t. \quad (12)$$

Using this, we can continue with our derivation, from (10), as follows

$$\begin{aligned}
\Delta\boldsymbol{\theta}^B_t &= \alpha\rho_t\delta_t\boldsymbol{\phi}_t \\
&\quad + \alpha\sum_{k=0}^{t-1}\left[\rho_k C^t_k\delta_t + (\rho_t - 1)\gamma_t\lambda_t\rho_k C^{t-1}_k\bar{\delta}^t_k\right]\boldsymbol{\phi}_k \\
&= \alpha\delta_t\left(\rho_t\boldsymbol{\phi}_t + \sum_{k=0}^{t-1}\rho_k C^t_k\boldsymbol{\phi}_k\right) \\
&\quad + (\rho_t - 1)\alpha\gamma_t\lambda_t\sum_{k=0}^{t-1}\rho_k C^{t-1}_k\bar{\delta}^t_k\boldsymbol{\phi}_k \\
&= \alpha\delta_t\boldsymbol{e}_t + (\rho_t - 1)\boldsymbol{u}_t, \quad (13)
\end{aligned}$$

where here we define an eligibility trace $\boldsymbol{e}_t \in \mathbb{R}^n$ (cf. Maei 2011, p. 71) as

$$\begin{aligned}
\boldsymbol{e}_t &= \rho_t\boldsymbol{\phi}_t + \sum_{k=0}^{t-1}\rho_k C^t_k\boldsymbol{\phi}_k \\
&= \rho_t\boldsymbol{\phi}_t + \gamma_t\lambda_t\rho_t\left(\rho_{t-1}\boldsymbol{\phi}_{t-1} + \sum_{k=0}^{t-2}\rho_k C^{t-1}_k\boldsymbol{\phi}_k\right) \\
&= \rho_t\left(\boldsymbol{\phi}_t + \gamma_t\lambda_t\boldsymbol{e}_{t-1}\right), \quad (14)
\end{aligned}$$

and a *provisional weight vector* $\boldsymbol{u}_t \in \mathbb{R}^n$ as

$$\begin{aligned}
\boldsymbol{u}_t &= \alpha\gamma_t\lambda_t\sum_{k=0}^{t-1}\rho_k C^{t-1}_k\bar{\delta}^t_k\boldsymbol{\phi}_k \\
&= \alpha\gamma_t\lambda_t\left[\sum_{k=0}^{t-2}\rho_k C^{t-1}_k\bar{\delta}^t_k\boldsymbol{\phi}_k + \rho_{t-1}C^{t-1}_{t-1}\bar{\delta}^t_{t-1}\boldsymbol{\phi}_{t-1}\right] \\
&= \alpha\gamma_t\lambda_t\left[\sum_{k=0}^{t-2}\rho_k C^{t-1}_k(\bar{\delta}^{t-1}_k + \bar{\delta}^t_{t-1})\boldsymbol{\phi}_k + \rho_{t-1}\bar{\delta}^t_{t-1}\boldsymbol{\phi}_{t-1}\right] \\
&= \alpha\gamma_t\lambda_t\left[\rho_{t-1}\gamma_{t-1}\lambda_{t-1}\sum_{k=0}^{t-2}\rho_k C^{t-2}_k\bar{\delta}^{t-1}_k\boldsymbol{\phi}_k\right. \\
&\qquad \left. + \sum_{k=0}^{t-2}\rho_k C^{t-1}_k\bar{\delta}^t_{t-1}\boldsymbol{\phi}_k + \rho_{t-1}\bar{\delta}^t_{t-1}\boldsymbol{\phi}_{t-1}\right] \\
&= \gamma_t\lambda_t\left[\rho_{t-1}\boldsymbol{u}_{t-1} + \alpha\bar{\delta}^t_{t-1}\boldsymbol{e}_{t-1}\right]. \quad (15)
\end{aligned}$$

Theorem 2 (Equivalence of PTD(λ))**.** *The PTD(λ) algorithm defined by (12–15), with* (2) *and* (4)*, is equivalent in the sense of* (8) *to the new forward view defined by (3–7).*

Proof. The result follows immediately from Theorem 1 and the above derivation. □

There are several observations to be made about the PTD(λ) algorithm vis-a-vis conventional TD(λ).

First, note that the first term of the PTD(λ) update (13) is $\alpha\delta_t\boldsymbol{e}_t$, the same as the complete update rule of TD(λ). The second term has a factor of $(\rho_t - 1)$ so, if ρ_t is always 1, as it is in the on-policy case, then this second term is always zero. This proves that PTD(λ) is identical to TD(λ) under on-policy training. This means that the interim forward view developed in Section 2 can also be viewed as an interim forward view for TD(λ) simply by taking the special case of $\rho_t = 1, \forall t$. The preceding derivation thus establishes that TD(λ) achieves an interim forward view.

Second, note that the per-time-step computational complexity of PTD(λ) is only slightly greater than that of TD(λ). TD(λ) maintains two n-component memory vectors, $\boldsymbol{\theta}_t$ and $\boldsymbol{e}_t$, while PTD maintains those and one more, the provisional weight vector $\boldsymbol{u}_t$. The updating of $\boldsymbol{u}_t$ in (15) and its use in (13) involve only a few vector–scalar multiplications and additions, while the eligibility-trace update (14) is unchanged from TD(λ). Thus the overall complexity of PTD(λ) remains of the same order as TD(λ).

By design, PTD(λ) with $\lambda = 1$ achieves equivalence to the forward view of off-policy Monte Carlo given in Figure 1, which we earlier suggested seemed impossible without greatly increased computational complexity. How does PTD(λ) square the circle? Recall the challenging case mentioned earlier of a trajectory that follows the target policy for a while and then deviates from it. Updates are made during the early part of the trajectory that then have to be precisely unmade upon deviation. What does PTD(λ) do in this scenario? In the first part of the trajectory, ρ_t will be 1, and thus the updates to $\boldsymbol{\theta}$ by (13) will be just like those of TD(λ). During this time, a near copy of these updates will be accumulating in $\boldsymbol{u}$. That is, according to (15), if γ_t and λ_t are near 1, and if $\rho_{t-1} = 1$ as is presumed in this example, then $\boldsymbol{u}_t$ will be incremented by $\alpha\bar{\delta}^t_{t-1}\boldsymbol{e}_{t-1}$, which is the same as the previous $\boldsymbol{\theta}$ update except for the slight difference between $\bar{\delta}^t_{t-1}$ and δ_{t-1}. Now suppose at time t there is a complete deviation from the target policy,

$\rho_t = 0$. By (13), $\boldsymbol{\theta}$ will be decremented by exactly $\boldsymbol{u}_t$, thus (approximately) undoing all the updates from the earlier part of the trajectory, which are now invalid. Also on this time step the new value $\boldsymbol{u}_{t+1}$ will start accumulating anew, from zero. In effect, the copy of the accumulated updates is removed from both $\boldsymbol{\theta}$ and $\boldsymbol{u}$. Of course, this is just an approximate characterization of what happens in PTD(λ). For example, there are small additional effects in the common cases when γ_t and λ_t are actually less than one, and there will be large additional effects if ρ_t is not less than 1, but greater (in this case the copy in $\boldsymbol{u}$ is *amplified*, and *adds to* $\boldsymbol{\theta}$). The online behavior of PTD(λ) is complex and subtly dependent on the values at each time step of γ_t, λ_t, and ρ_t. This is why it is important that we can understand PTD(λ) in terms of high-level goals (the forward view of Section 2), and that its precise form has been derived explicitly to achieve these goals (this section).

So far we have validated the way importance sampling correction is done in PTD(λ) by pointing out that, if $\lambda_t = 1\ \forall t$, then it achieves the exact same updates as the Monte Carlo forward view in Figure 1. But how do we know that this Monte Carlo forward view is correct, and what about other values of λ_t? Importance sampling corrections are meant to "correct" for the data being due to the behavior policy, b, rather than to the target policy, π. Of course, the actual updates are not the same because the data is not the same; the idea is that the expected updates can be made the same. That is, we would like PTD(λ)'s expected update for any state, with actions from any behavior policy, to equal the expected update of TD(λ) for the same state with actions from the target policy. The following theorem establishes a slightly stronger result, that the forward-view errors are the same in expected value. The expected updates are then equal because they are this error times the feature vector at the time (and the step-size parameter). Recall that the forward-view error of TD(λ) is the same as $\delta^{\lambda\rho}_{k,t}$ except with all $\rho_t = 1\ \forall t$, which we here denote as $\delta^{\lambda 1}_{k,t}$.

Theorem 3 (On-policy and off-policy expectations)**.** *For any state s,*

$$\mathbb{E}_b\left[\delta^{\lambda\rho}_{k,t}\middle| S_k = s\right] = \mathbb{E}_\pi\left[\delta^{\lambda 1}_{k,t}\middle| S_k = s\right], \tag{16}$$

where $\mathbb{E}_b$ and $\mathbb{E}_\pi$ denote expectations under the behavior and target policies, and $\delta^{\lambda 1}_{k,t}$ denotes a version of $\delta^{\lambda\rho}_{k,t}$ with $\rho_t = 1, \forall t$. (Proved in the supplementary material.)

This result establishes the off-line equivalence in expectation of PTD(λ) and TD(λ), and of our Monte Carlo forward-view and TD(1). The latter is significant because it is already well known that TD(1) is equivalent in this sense to a valid on-policy Monte Carlo algorithm (Sutton & Barto 1998, Chapter 7). We have thus also validated the way importance sampling correction is done in our off-policy Monte Carlo forward view (Figure 1).

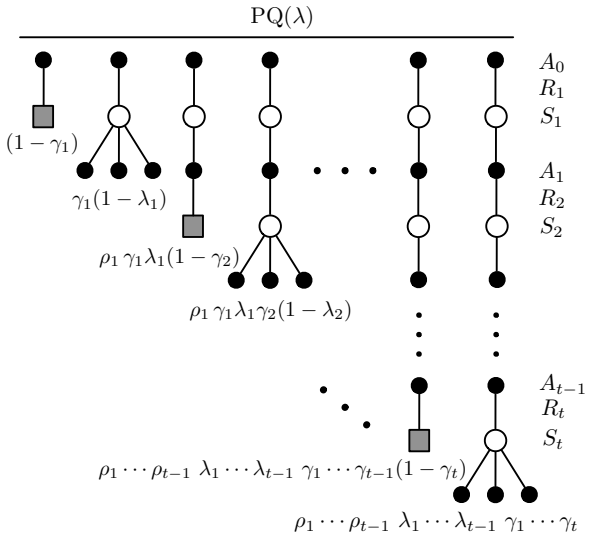

Figure 3. The backup diagram and forward view of PQ(λ).

5. PQ(λ)

In this section we apply the interim-equivalence technique to derive an action-value algorithm—a new version of Q(λ) with a Monte Carlo equivalence similar to that which we have just shown for PTD(λ). The learning problem is now to find a weight vector $\boldsymbol{\theta}$ such that:

$$\boldsymbol{\theta}^\top\boldsymbol{\phi}(s,a) \approx \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} R_{t+1}\prod_{k=1}^{t}\gamma(S_k)\middle| S_0 = s, A_0 = a\right], \tag{17}$$

where $\boldsymbol{\phi}(s,a)$ is a feature vector for the state–action pair. The intuition for the forward view is given in Figure 3. It is similar to the state-value case (Figure 2), except that trajectories now start with state–action pairs and, if they end with bootstrapping, an average is done over the final action values weighted by the target policy. That is, the final correction term is not $\boldsymbol{\theta}^\top\boldsymbol{\phi}(S_t)$, but $\boldsymbol{\theta}^\top\bar{\boldsymbol{\phi}}^\pi_t$, where:

$$\bar{\boldsymbol{\phi}}^\pi_t = \sum_a \pi(a|S_t)\boldsymbol{\phi}(S_t,a). \tag{18}$$

Our previous notation is redefined for action values as:

$$\boldsymbol{\phi}_t = \boldsymbol{\phi}(S_t, A_t) \tag{19}$$

$$\epsilon^t_k = R_{k+1} + R_{k+2} + \cdots + R_t - \boldsymbol{\theta}^\top\boldsymbol{\phi}_k \tag{20}$$

$$\bar{\delta}^t_k = \epsilon^t_k + \boldsymbol{\theta}^\top\bar{\boldsymbol{\phi}}^\pi_t \tag{21}$$

$$\delta_t = \epsilon^{t+1}_t + \gamma_{t+1}\boldsymbol{\theta}^\top\bar{\boldsymbol{\phi}}^\pi_{t+1} \tag{22}$$

Using these, and repeating the steps from Section 2, the forward-view error is redefined for action values as

$$\delta^{\lambda\rho}_{k,t} = \sum_{i=k+1}^{t-1} C^{i-1}_k\left[(1-\gamma_i)\epsilon^i_k + \gamma_i(1-\lambda_i)\bar{\delta}^i_k\right] + C^{t-1}_k\left[(1-\gamma_t)\epsilon^t_k + \gamma_t\bar{\delta}^t_k\right], \tag{23}$$

which again can be written recursively, for $k < t$:

$$\delta^{\lambda\rho}_{k,t+1} = \delta^{\lambda\rho}_{k,t} + C^t_k \delta_t + C^t_k \boldsymbol{\theta}^\top (\boldsymbol{\phi}_t - \bar{\boldsymbol{\phi}}^\pi_t) + (\rho_t - 1)\gamma_t \lambda_t C^{t-1}_k \bar{\delta}^t_k, \tag{24}$$

as we show in the supplementary material.

With the new notation, the forward-view update is the same as for PTD(λ): $\Delta\boldsymbol{\theta}^F_{k,t} = \alpha \delta^{\lambda\rho}_{k,t} \boldsymbol{\phi}_k$ (7). Applying the interim equivalence technique, as before, again brings us to (10), and then to an efficient implementation:

$$\begin{aligned}
\Delta\boldsymbol{\theta}^B_t &= \alpha \delta^{\lambda\rho}_{t,t+1} \boldsymbol{\phi}_t + \alpha \sum_{k=0}^{t-1} \left[\delta^{\lambda\rho}_{k,t+1} - \delta^{\lambda\rho}_{k,t} \right] \boldsymbol{\phi}_k \qquad (10) \\
&= \alpha \delta_t \boldsymbol{\phi}_t + \alpha \sum_{k=0}^{t-1} C^t_k \delta_t \boldsymbol{\phi}_k + \alpha \sum_{k=0}^{t-1} C^t_k \boldsymbol{\theta}^\top (\boldsymbol{\phi}_t - \bar{\boldsymbol{\phi}}^\pi_t) \boldsymbol{\phi}_k \\
&\quad + \alpha (\rho_t - 1) \gamma_t \lambda_t \sum_{k=0}^{t-1} C^{t-1}_k \bar{\delta}^t_k \boldsymbol{\phi}_k \\
&= \alpha \delta_t \left(\boldsymbol{\phi}_t + \sum_{k=0}^{t-1} C^t_k \boldsymbol{\phi}_k \right) + \alpha \boldsymbol{\theta}^\top (\boldsymbol{\phi}_t - \bar{\boldsymbol{\phi}}^\pi_t) \sum_{k=0}^{t-1} C^t_k \boldsymbol{\phi}_k \\
&\quad + (\rho_t - 1) \alpha \gamma_t \lambda_t \sum_{k=0}^{t-1} C^{t-1}_k \bar{\delta}^t_k \boldsymbol{\phi}_k \\
&= \alpha \delta_t \boldsymbol{e}_t + \alpha \boldsymbol{\theta}^\top (\boldsymbol{\phi}_t - \bar{\boldsymbol{\phi}}^\pi_t)(\boldsymbol{e}_t - \boldsymbol{\phi}_t) + (\rho_t - 1) \boldsymbol{u}_t \qquad (25)
\end{aligned}$$

where $\boldsymbol{e}_t \in \mathbb{R}^d$ and $\boldsymbol{u}_t \in \mathbb{R}^d$ are again eligibility-trace and provisional-weight vectors, but defined and updated slightly differently for the action-value case:

$$\begin{aligned}
\boldsymbol{e}_t &= \boldsymbol{\phi}_t + \sum_{k=0}^{t-1} C^t_k \boldsymbol{\phi}_k = \boldsymbol{\phi}_t + \gamma_t \lambda_t \rho_t \left[\sum_{k=0}^{t-2} C^{t-1}_k \boldsymbol{\phi}_k + \boldsymbol{\phi}_{t-1} \right] \\
&= \boldsymbol{\phi}_t + \gamma_t \lambda_t \rho_t \boldsymbol{e}_{t-1}, \qquad (26)
\end{aligned}$$

(as in Maei & Sutton 2010) and

$$\begin{aligned}
\boldsymbol{u}_t &= \alpha \gamma_t \lambda_t \sum_{k=0}^{t-1} C^{t-1}_k \bar{\delta}^t_k \boldsymbol{\phi}_k \\
&= \gamma_t \lambda_t \Big(\rho_{t-1} \boldsymbol{u}_{t-1} + \alpha \bar{\delta}^t_{t-1} \boldsymbol{e}_{t-1} \\
&\quad + \alpha \boldsymbol{\theta}^\top \left(\boldsymbol{\phi}_{t-1} - \bar{\boldsymbol{\phi}}^\pi_{t-1} \right) \left(\boldsymbol{e}_{t-1} - \boldsymbol{\phi}_{t-1} \right) \Big). \qquad (27)
\end{aligned}$$

The derivation of (27) is provided in the supplementary material.

Theorem 4 (Equivalence of PQ(λ))**.** *The PQ(λ) algorithm defined by (25–27), with (18–22) and* (2)*, is equivalent in the sense of* (8) *to the forward view* (7) *and* (23) *with* (5)*.*

Proof. The result follows immediately from Theorem 1 and the above derivation. □

When the target policy is greedy with respect to the action value function, then PQ(λ) can be considered a version of Q(λ). In this case, $\boldsymbol{\theta}^\top \bar{\boldsymbol{\phi}}^\pi_t = \max_a \boldsymbol{\theta}^\top \boldsymbol{\phi}(S_t, a)$, and one-step Q-learning can be written simply as $\alpha \delta_t \boldsymbol{\phi}_t$. PQ($\lambda$) with $\lambda = 0$ is exactly equivalent to this because the second and third terms of (25) are zero ($\boldsymbol{e}_t = \boldsymbol{\phi}_t$ and $\boldsymbol{u}_t = 0$). Even if $\lambda > 0$, the second term in (25) is zero because either a greedy action is taken, in which case $\boldsymbol{\phi}_t = \bar{\boldsymbol{\phi}}^\pi_t$, or else a non-greedy action is taken, in which case $\rho_t = 0$ and hence $\boldsymbol{e}_t = \boldsymbol{\phi}_t$. Without the second term, the update is like that of Watkin's Q(λ) except that the eligibility trace includes a factor of ρ_t, and there's an additional term of $(\rho_t - 1)\boldsymbol{u}_t$. The ρ_t is key to maintaining the validity of the expectations (as in Theorem 3), and the additional term is responsible for 'un-making' the past updates when a non-greedy action is taken—PQ(λ)'s signature feature which makes it equivalent to Monte Carlo (for any target policy) when $\lambda = 1$. Note that PQ(λ) remains of linear complexity—of the same order as, for example, Watkins's Q(λ) with linear function approximation.

In the on-policy case ($\rho_t = 1$), PQ(λ) becomes the "summation Q(λ)" algorithm of Rummery (1995), which might also be called Expected Sarsa(λ), as it is an eligibility-trace extension of the Expected Sarsa algorithm investigated by van Seijen, van Hasselt, Whiteson, and Wiering (2009).

6. Conclusions, limitations, and future work

We have presented a new technique for deriving learning algorithms and used it to derive new off-policy versions of TD(λ) and Q(λ). The new algorithms are the first off-policy algorithms to become equivalent to Monte Carlo algorithms when $\lambda = 1$. Our formal results have treated the case of linear function approximation and of fixed target and behavior policies. Convergence in the tabular case should follow by simple application of existing results. In the case of linear function approximation, a gradient-based extension (a la Maei 2011) is required to obtain convergence guarantees (as in van Hasselt et al., 2014). Convergence in the control case has never been established for any algorithm for $\lambda > 0$ and remains an open problem. Another intriguing direction for future work is to extend our interim equivalence technique under off-line updating to derive exact equivalences under online updating, using the ideas recently introduced (for on-policy TD(λ)) by van Seijen and Sutton (2014).

Acknowledgements

The authors benefited greatly from discussions with Harm van Seijen and Patrick Pilarski. This work was supported by Alberta Innovates – Technology Futures, NSERC, and the Alberta Innovates Centre for Machine Learning.

References

Downey, C., Sanner, S. (2010). Temporal difference Bayesian model averaging: A Bayesian perspective on adapting lambda. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 311–318.

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

Kearns, M. J., Singh, S. P. (2000). Bias-variance error bounds for temporal difference updates. In *Proceedings of the 13th Annual Conference on Computational Learning Theory*, pp. 142–147.

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

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.

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

Peng, J. (1993). *Efficient Dynamic Programming-Based Learning for Control*. PhD thesis, Northeastern University, Boston.

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.

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

Rummery, G. A. (1995). *Problem Solving with Reinforcement Learning*. PhD thesis, Cambridge University.

Singh, S. P., Dayan, P. (1998). Analytical mean squared error curves for temporal difference learning. *Machine Learning 32*:5–40.

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

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

Sutton, R. S. (1995). TD models: Modeling the world at a mixture of time scales. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 531–539. Morgan Kaufmann.

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

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 Tenth International Conference on Autonomous Agents and Multiagent Systems*, Taipei, Taiwan.

Sutton, R. S., Singh, S. (1994). On bias and step size in temporal-difference learning. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pp. 91–96, New Haven, CT. Yale University.

van Hasselt, H., Mahmood, A. R., Sutton, R. S. (2014). Off-policy TD(λ) with a true online equivalence. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, Quebec City, Canada.

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

van Seijen, H., van Hasselt, H., Whiteson, S., Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa. In *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 177–184.

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

Yu, H. (2010). Convergence of least-squares temporal difference methods under general conditions. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 1207-1214.