RL Ch12 Eligibility Traces

Le-Rong Hsu

July 2023

1 The λ-return

λ can be viewed as the **balance between Monte Carlo methods and one-step TD methods**.

The general form of n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}) \tag{12.1}$$

for $0 \leq t < T - n$. The TD(λ) algorithm can be understood as one **particular way of averaging n-step updates**. This average contains all the n-step updates, each weighted proportionally to λ^{n-1}. The λ-return is defined by

$$G_t^{\lambda} \doteq (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{12.2}$$

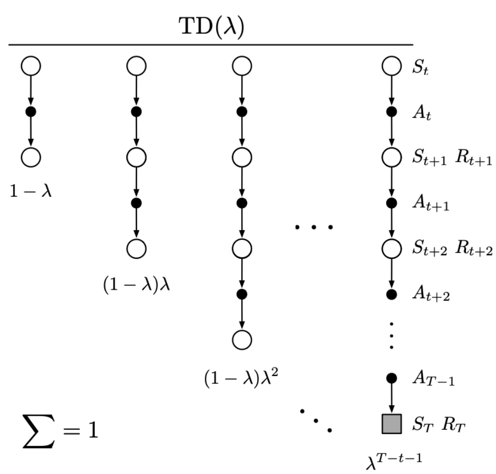


Figure 12.1 $\lambda = 0 \rightarrow$ one-step TD, $\lambda = 1 \rightarrow$ Monte Carlo

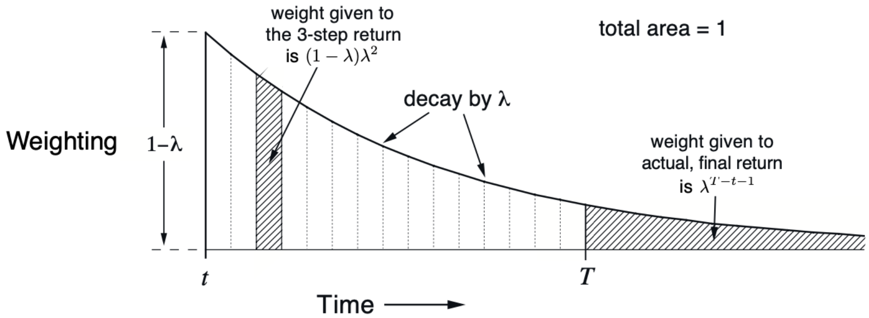


Figure 12.2 Weighting of λ-return

We also write $G^{\lambda}_{t:t+n}$ as

$$G^{\lambda}_{t} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \tag{12.3}$$

Remark. The parameter λ characterizes how fast the exponential weighting in Figure 12.2 falls off.

λ-return target:

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \alpha[G^{\lambda}_t - \hat{v}(S_t, \mathbf{w_t})]\nabla\hat{v}(S_t, \mathbf{w_t})$$

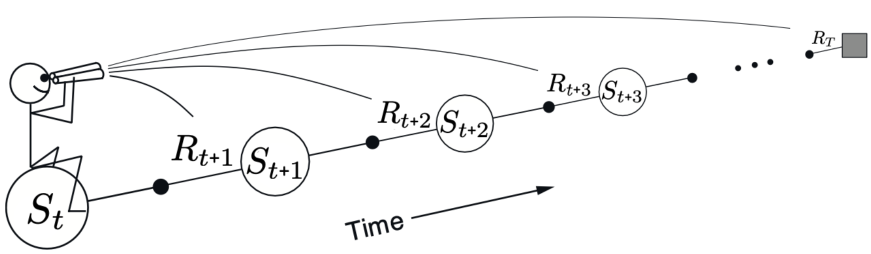


Figure 12.4 λ determines how to combine the future rewards

2 TD(λ)

TD(λ) improves over the off-line λ-return algorithm in three ways.

- it updates the weight vector on every step of an episode
- its computations are equally distributed in time
- it can be applied to continuing problems

Weight Vector vs. Eligibility Trace

- long-term memory, accumulating over the lifetime of the system
- short-term memory, typically lasting less time than the length of an episode

TD(λ)

$$\begin{aligned}\mathbf{z}_{-1} &= 0 \\ \mathbf{z}_{t+1} &= \gamma\lambda\mathbf{z}_t + \nabla\hat{v}(S_t, \mathbf{w}_t), 0 \le t \le T \end{aligned} \tag{12.5}$$

The eligibility trace keeps track of which components of the weight vector have contributed to recent (in terms of $\gamma\lambda$) state valuations.

The TD error for state-value prediction is

$$\delta_t \doteq R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{12.6}$$

The weight vector is updated on each step

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t. \tag{12.7}$$

Remarks on TD(λ)

- If $\lambda = 0$, then $\mathbf{z}_{t+1} = \nabla\hat{v}(S_t, \mathbf{w}_t)$ and it's semi-gradient TD(0) in Ch9.
- If $\lambda = 1$, then it's undiscounted episodic task.
- If $\lambda = 1$, the algorithm is also called TD(1).
- The choice of λ affects the performance.

Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)
  Loop for each step of episode:
  | Choose $A \sim \pi(\cdot|S)$
  | Take action A, observe R, S'
  | $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S,\mathbf{w})$
  | $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
  | $S \leftarrow S'$
  until S' is terminal

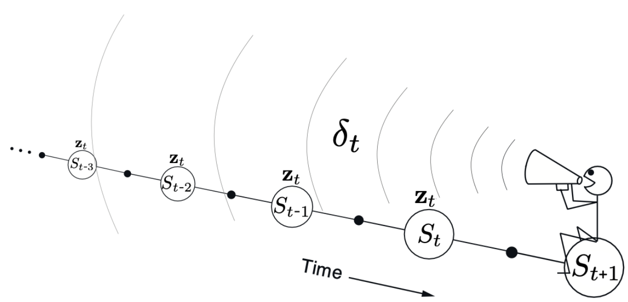


Figure 12.5 Each update depends on the current TD error combined with the current eligibility traces of past events.

Convergence

Linear TD(λ) has been proved to converge in the on-policy case according to (2.7). Convergence is not to the minimum-error weight vector, but to **a nearby weight vector** that depends on λ.

For continuing case,

$$\overline{\text{VE}}(\mathbf{w}_\infty) \le \frac{1-\gamma\lambda}{1-\gamma} \min_{\mathbf{w}} \overline{\text{VE}}(\mathbf{w}). \tag{12.8}$$

As $\lambda \to 1$, the bound approaches the minimum error. In practice, **this is a poor choice**.

3 n-step Truncated λ-return Methods

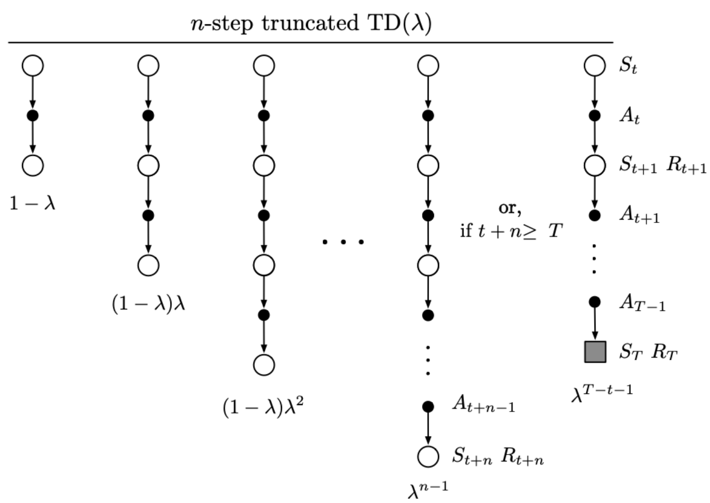
Motivation

- λ-return $G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$ is unknown until the end of the episode
- In continuing case, λ-return is unknown as it depends on n-steps returns for arbitrarily large n.
- Dependence becomes weaker after some time step.

Truncated TD(λ)

$$G_{t:h}^\lambda = (1-\lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_{t:h} \tag{12.9}$$

for $0 \leq t < h \leq T$. The truncated λ-return immediately gives rise to a family of n-step λ-return algorithms similar to the n-step methods of Chapter 7.



$$\mathbf{w}_{t+n} = \mathbf{w}_{t+n-1} + \alpha[G_{t:t+n}^\lambda - \hat{v}(S_t, \mathbf{w}_{t+n-1})] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), 0 \leq t < T.$$

No updates are made on the first $n-1$ time steps of each episode, and $n-1$ additional updates are made upon termination. Hence efficient implementation (why?):

$$G_{t:t+k}^\lambda = \hat{v}(S_t, \mathbf{w}_{t-1}) + \sum_{i=t}^{t+k-1} (\gamma\lambda)^{i-t} \delta_i' \tag{12.10}$$

$$\delta_i' \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_{t-1})$$

4 Redoing Updates: Online λ-return Algorithm

- updates during the episodes
- actions are determined by current value estimates

Trade off

n should be large so that the method closely approximates the off-line λ-return algorithm, but be large so that the method closely approximates the off-line λ-return algorithm.

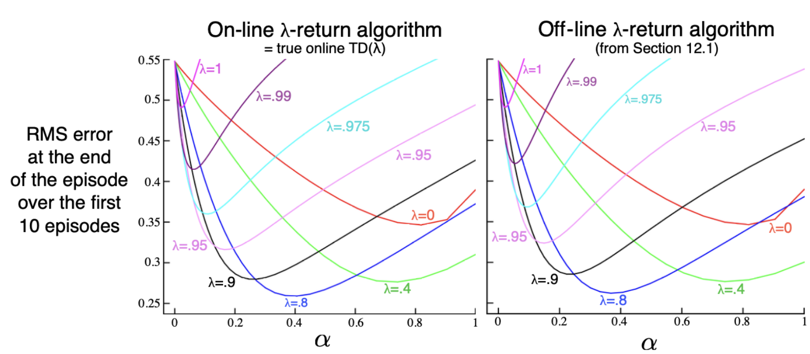
Online λ-return

The idea of online λ-return is very simple: at each time step, the truncated λ-returns from all previous time steps are updated, such that they are now **truncated at the current time step**.

$$
\begin{aligned}
\mathbf{w}_0^0 : & \mathbf{w}_0^0 = \mathbf{w}_{\text{init}} \\
\mathbf{w}_1^1 : & \mathbf{w}_0^1 = \mathbf{w}_{\text{init}} \\
& \mathbf{w}_1^1 = \mathbf{w}_0^1 + \alpha[G_{1:0}^\lambda - \hat{v}(S_0, \mathbf{w}_0^1)]\nabla\hat{v}(S_0, \mathbf{w}_0^1) \\
\mathbf{w}_2^2 : & \mathbf{w}_0^2 = \mathbf{w}_{\text{init}} \\
& \mathbf{w}_1^2 = \mathbf{w}_0^2 + \alpha[G_{2:0}^\lambda - \hat{v}(S_0, \mathbf{w}_0^2)]\nabla\hat{v}(S_0, \mathbf{w}_0^2) \\
& \mathbf{w}_2^2 = \mathbf{w}_1^2 + \alpha[G_{2:1}^\lambda - \hat{v}(S_1, \mathbf{w}_1^2)]\nabla\hat{v}(S_1, \mathbf{w}_1^2) \\
& \vdots
\end{aligned}
$$

The general form for the update is

$$\mathbf{w}_{t+1}^h = \mathbf{w}_t^h + \alpha[G_{t:h}^\lambda - \hat{v}(S_t, \mathbf{w}_t^h)]\nabla\hat{v}(S_t, \mathbf{w}_t^h)), \quad 0 \le t < h \le T.$$



5 True Online TD(λ)

- Find a compact, efficient way of computing each $\mathbf{w}_t^t$ from the one before

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t + \alpha(\mathbf{w}_t^T\mathbf{x}_t - \mathbf{w}_{t-1}^T\mathbf{x}_t)(\mathbf{z}_t - \mathbf{x}_t)$$

where $\mathbf{x}_t = \mathbf{x}(S_t)$ and $\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)$. $\mathbf{z}_t$ is defined by

$$\mathbf{z}_{t+1} = \gamma\lambda\mathbf{z}_t - (1 - \alpha\gamma\lambda\mathbf{z}_{t-1}^T\mathbf{x}_t)\mathbf{x}_t \tag{12.11}$$

This algorithm has been proven to produce exactly the same sequence of weight

True online TD(λ) for estimating $\mathbf{w}^\top\mathbf{x} \approx v_\pi$

Input: the policy π to be evaluated
Input: a feature function $\mathbf{x} : \mathcal{S}^+ \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize state and obtain initial feature vector $\mathbf{x}$
  $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)
  $V_{old} \leftarrow 0$ (a temporary scalar variable)
  Loop for each step of episode:
  | Choose $A \sim \pi$
  | Take action A, observe R, $\mathbf{x}'$ (feature vector of the next state)
  | $V \leftarrow \mathbf{w}^\top\mathbf{x}$
  | $V' \leftarrow \mathbf{w}^\top\mathbf{x}'$
  | $\delta \leftarrow R + \gamma V' - V$
  | $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\mathbf{x}$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + V - V_{old})\mathbf{z} - \alpha(V - V_{old})\mathbf{x}$
  | $V_{old} \leftarrow V'$
  | $\mathbf{x} \leftarrow \mathbf{x}'$
  until $\mathbf{x}' = \mathbf{0}$ (signaling arrival at a terminal state)

vectors for $0 \le t \le T$ as the online λ-return algorithm. However, the algorithm is much less expensive.

- memory: same as TD(λ)
- time complexity: 50% more than TD(λ)

Different Traces

- accumulating trace:

$$\begin{aligned} &\mathbf{z}_{-1} = 0 \\ &\mathbf{z}_{t+1} = \gamma\lambda\mathbf{z}_t + \nabla\hat{v}(S_t, \mathbf{w}_t), 0 \le t \le T \end{aligned} \tag{12.5}$$

- dutch trace: $\mathbf{z}_{t+1} = \gamma\lambda\mathbf{z}_t - (1 - \alpha\gamma\lambda\mathbf{z}_{t-1}^T\mathbf{x}_t)\mathbf{x}_t$

- replacing trace:

$$z_{i:t} = \begin{cases} 1 & \text{if } x_{i:t} = 1 \\ \gamma\lambda z_{i:t-1} & \text{otherwise.} \end{cases}$$

Dutch traces usually perform better than replacing traces and have a clearer theoretical basis. **Accumulating traces** remain of interest for **nonlinear function approximations** where dutch traces are not available.

6 Sarsa(λ)

Recall the n-step return of action value:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}) t + n < T.$$

with $G_{t:t+n} \doteq G_t$ for $t + n \geq T$. The action-value form of the off-line λ-return algorithm

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \alpha[G_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w_t})] \nabla \hat{q}(S_t, A_t, \mathbf{w_t}) \tag{12.15}$$

where $G_t^\lambda \doteq G_\infty^\lambda$. The update rule is the same as TD(λ):

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t$$

TD error:

$$\delta_t \doteq R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \tag{12.16}$$

Eligibility trace:

$$\begin{aligned} &\mathbf{z}_{-1} = 0 \\ &\mathbf{z}_{t+1} = \gamma\lambda \mathbf{z}_t + \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \quad 0 \leq t \leq T. \end{aligned}$$

Sarsa(λ) with binary features and linear function approximation for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a function $\mathcal{F}(s,a)$ returning the set of (indices of) active features for s, a
Input: a policy π
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$, small $\varepsilon > 0$
Initialize: $\mathbf{w} = (w_1, \ldots, w_d)^\top \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$), $\mathbf{z} = (z_1, \ldots, z_d)^\top \in \mathbb{R}^d$

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
  $\mathbf{z} \leftarrow \mathbf{0}$
  Loop for each step of episode:
    Take action A, observe R, S'
    $\delta \leftarrow R$
    Loop for i in $\mathcal{F}(S, A)$:
      $\delta \leftarrow \delta - w_i$
      $z_i \leftarrow z_i + 1$ (accumulating traces)
      or $z_i \leftarrow 1$ (replacing traces)
    If S' is terminal then:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
      Go to next episode
    Choose $A' \sim \pi(\cdot|S')$ or ε-greedy according to $\hat{q}(S', \cdot, \mathbf{w})$
    Loop for i in $\mathcal{F}(S', A')$: $\delta \leftarrow \delta + \gamma w_i$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z}$
    $S \leftarrow S'; A \leftarrow A'$

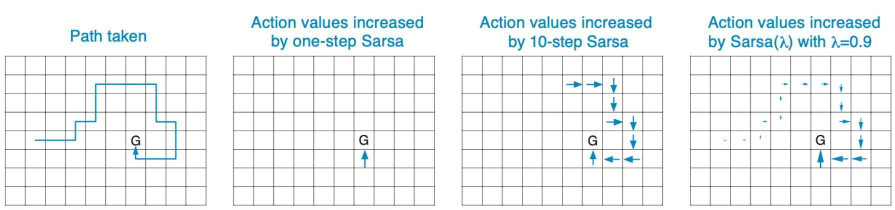


Figure 12.1 Traces in Gridworld

Conclusion:

- Eligibility trace method would update all the action values up to the beginning of the episode.
- The fading strategy (e.g. eligibility trace method) is often the best.

True online Sarsa(λ) for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$, small $\varepsilon > 0$
Initialize: $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
  $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
  $\mathbf{z} \leftarrow \mathbf{0}$
  $Q_{old} \leftarrow 0$
  Loop for each step of episode:
  | Take action A, observe R, S'
  | Choose $A' \sim \pi(\cdot|S')$ or ε-greedy according to $\hat{q}(S', \cdot, \mathbf{w})$
  | $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
  | $Q \leftarrow \mathbf{w}^\top \mathbf{x}$
  | $Q' \leftarrow \mathbf{w}^\top \mathbf{x}'$
  | $\delta \leftarrow R + \gamma Q' - Q$
  | $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\mathbf{x}$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x}$
  | $Q_{old} \leftarrow Q'$
  | $\mathbf{x} \leftarrow \mathbf{x}'$
  | $A \leftarrow A'$
  until S' is terminal

Summary comparison

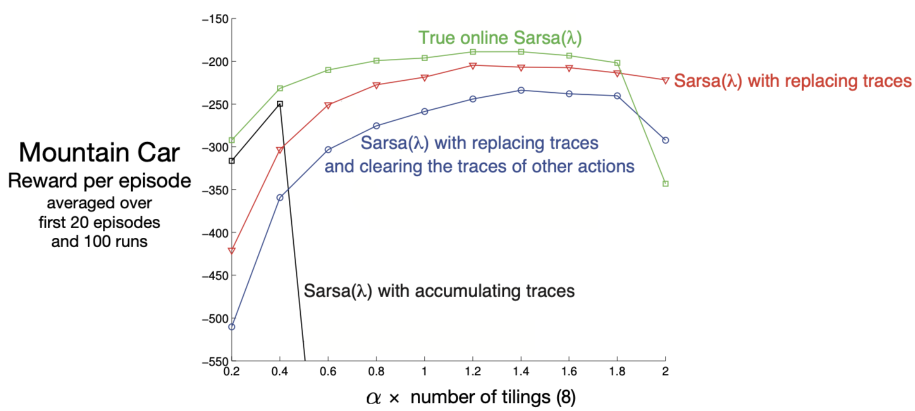


Figure 12.11 True online Sarsa(λ) performed better than regular Sarsa(λ)

7 Implementation Issues

- Typical values of γ and λ the eligibility traces of almost all states are almost always nearly zero
- Only those states that have been **recently visited** will have traces significantly greater than zero. Only these few states need to be updated to closely approximate these algorithms.
- We only need to keep track of and update **only the few traces that are significantly greater than zero**.
- Eligibility traces generally cause only a **doubling of the required memory** and computation per step if DNN and backpropagation are used.
- **Truncated** λ-return methods can be **computationally efficient** though additional memory is required.