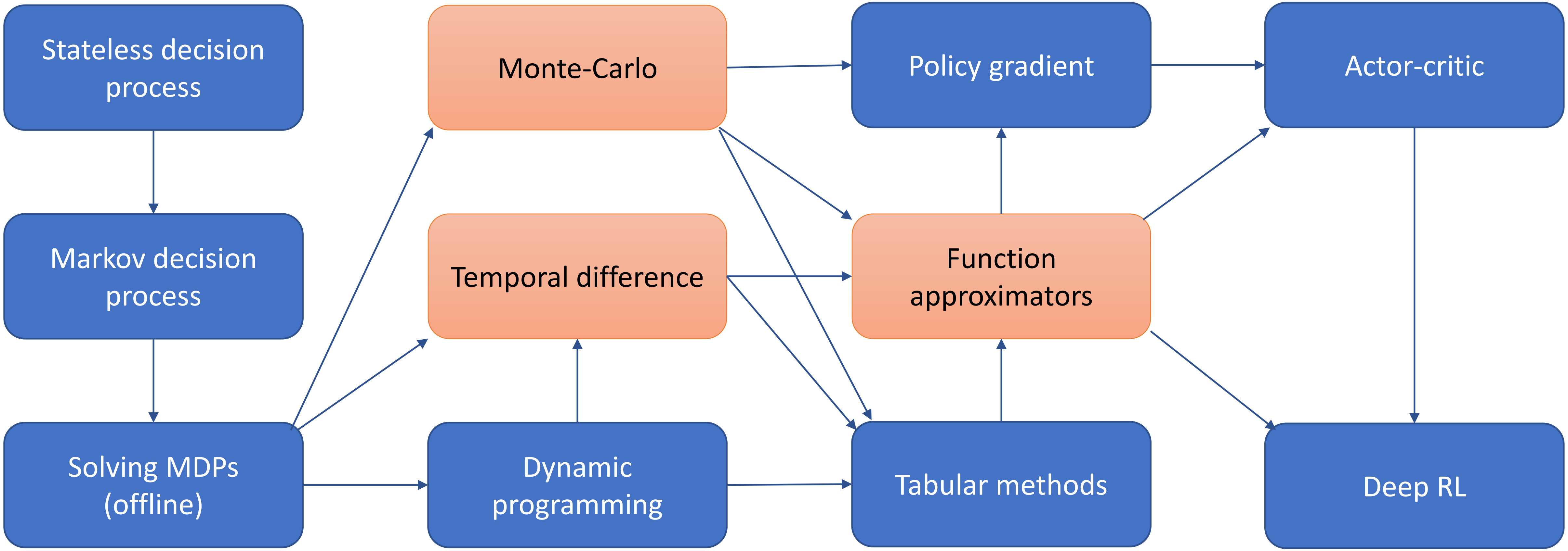
CSCE-642 Reinforcement Learning
Ch 12: Eligibility Traces

Instructor: Guni Sharon

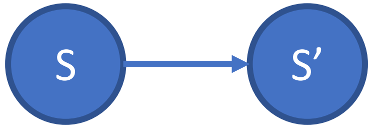
CSCE-689, Reinforcement Learning



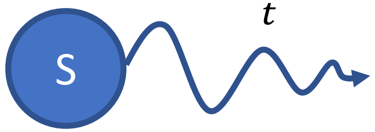
So far…

- TD(0) – propagate state/action values from immediate neighbor/s
  - Q-learning, SARSA
  - Biased samples
- Monte Carlo TD(1) - update state/action value according to return from a full episode
  - High variance samples
  - On policy
- Combine the two to balance bias and variance
  - n-step return

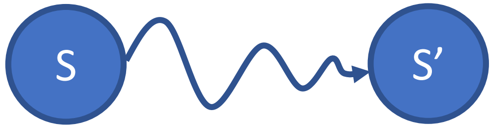
$$\text{target } v(s) = R + \hat{v}(S')$$



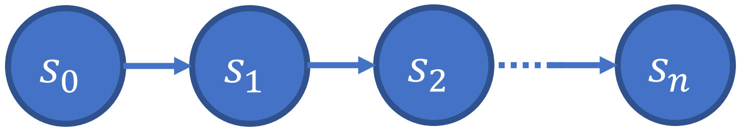
$$\text{target } v(s) = \sum_t R_t$$



$$\text{target } v(s) = \hat{v}(S') + \sum_{t:n} R_t$$

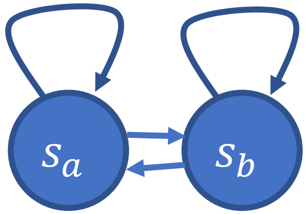


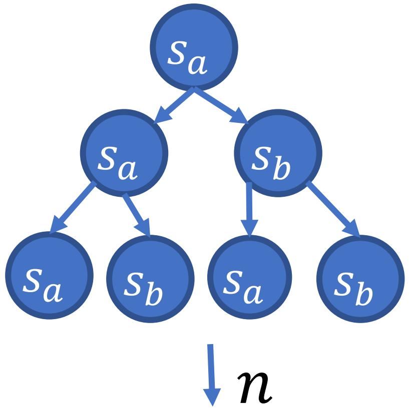
n-step return intuition



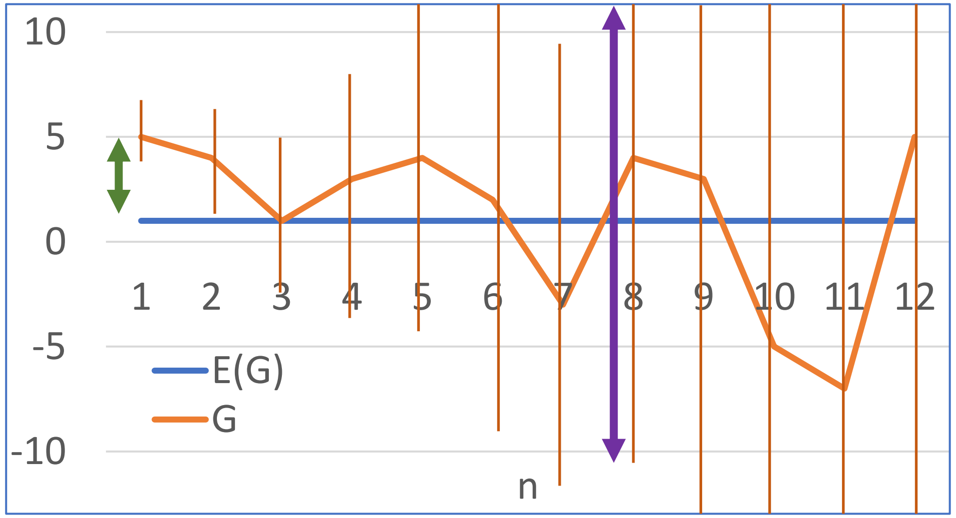
- **Goal**: estimate the value of s_0 while balancing bias and variance
- Estimating $v(s_0)$ based on 1-step TD, i.e., $v(s_0) = R_1 + \gamma\hat{v}(s_1)$
  - High bias, low variance
- Estimating $v(s_0)$ based on 2-step TD i.e., $v(s_0) = r_1 + \gamma r_2 + \gamma^2\hat{v}(s_2)$
  - Lower bias (we observe the true rewards), higher variance (the probability of reaching s_2 is lower than that of reaching s_1)
- Estimating $v(s_0)$ based on n-step TD i.e., $v(s_0) = \gamma^n\hat{v}(s_n) + \sum_{t=1:n} \gamma^{t-1} r_t$
  - Even lower bias (we observe the true rewards), even higher variance (the probability of following this exact trajectory diminishes (exponentially) with *n*

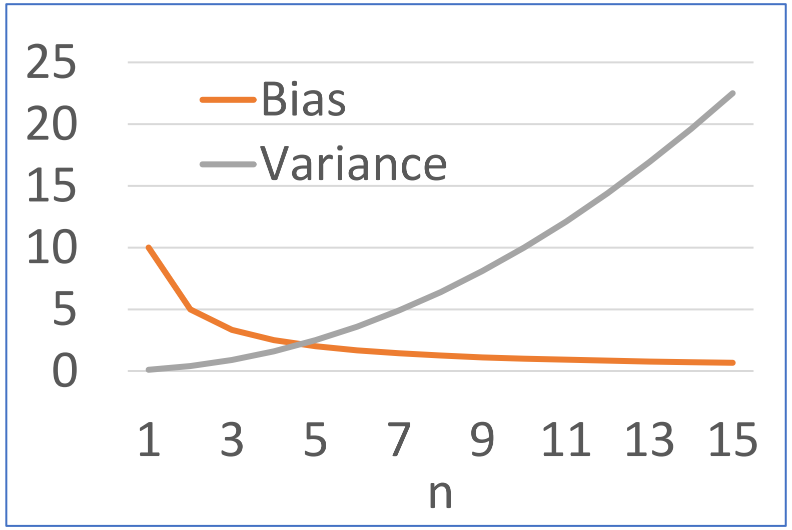
n-step return intuition



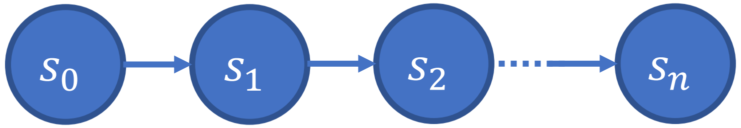


- Assume that every state, action (S, A) pair leads to one of two outcomes (S'_a, S'_b) with equal probabilities
  - What is the probability of sampling a specific 2 step trajectory (S_0, S_1, S_2)?
  - What is the probability of sampling a specific n step trajectory?
  - Trajectory probability diminishes exponentially with n
- Goal: find n that optimizes some combination of bias and variance





λ return

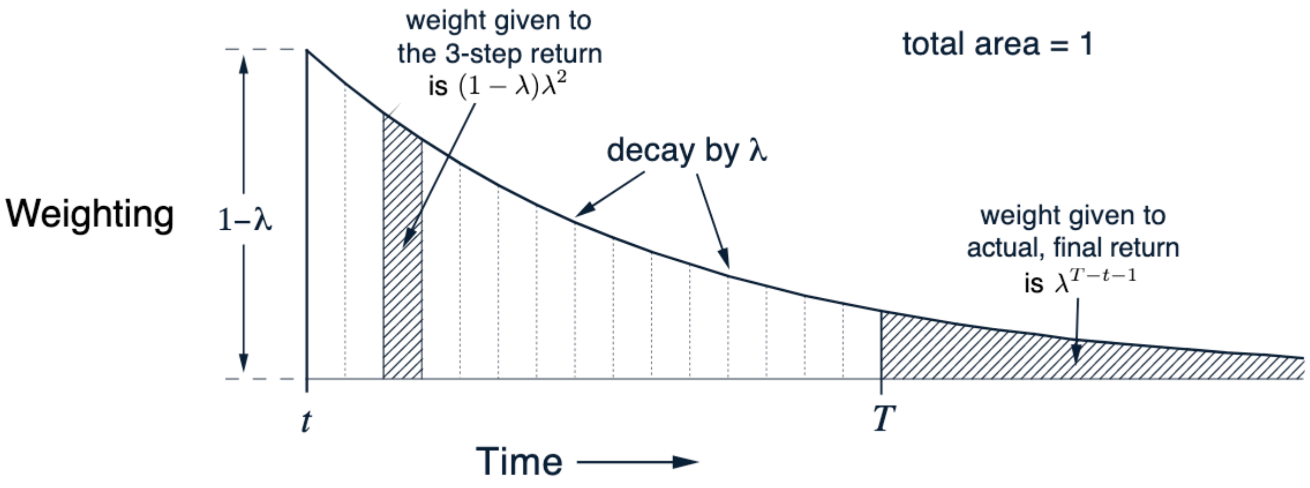


- Consider the n-step return for every possible value of n
  - $G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \hat{v}(S_{t+n})$
- Assign an exponentially decaying (with n) weight for each such return
  - Representing the decaying likelihood of the sampled trajectory
  - Weight(n) $= (1-\lambda)\lambda^{n-1}$ where $0 < \lambda < 1$ is a hyper-parameter
- Adjusted return: $G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$
- All weights sum to one, $\sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1} = 1$
- Gives a weighted average over all n-step returns

Bounded horizon (T) adjustments

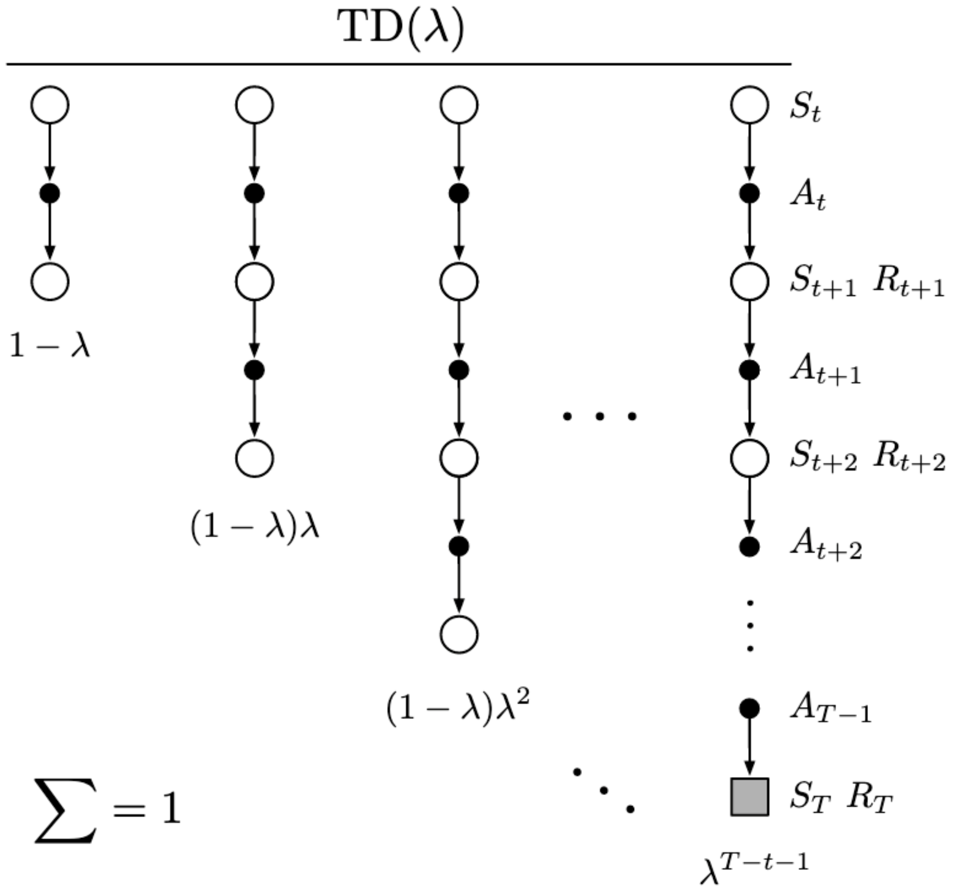
- $G_{t:T}^{\lambda} = (\lambda^{T-t-1} + (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1})G_{t:t+n}$
- All weights sum to 1: $\lambda^{T-t-1} + (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} = 1$

- Recall that:
  - $\sum_{n=1}^{T} ax^{n-1} = \frac{a(1-x^T)}{1-x}$
  - So:
  - $\lambda^{T-t-1} + (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1}$
  - $= \lambda^{T-t-1} + \frac{(1-\lambda)(1-\lambda^{T-t-1})}{1-\lambda} = 1$



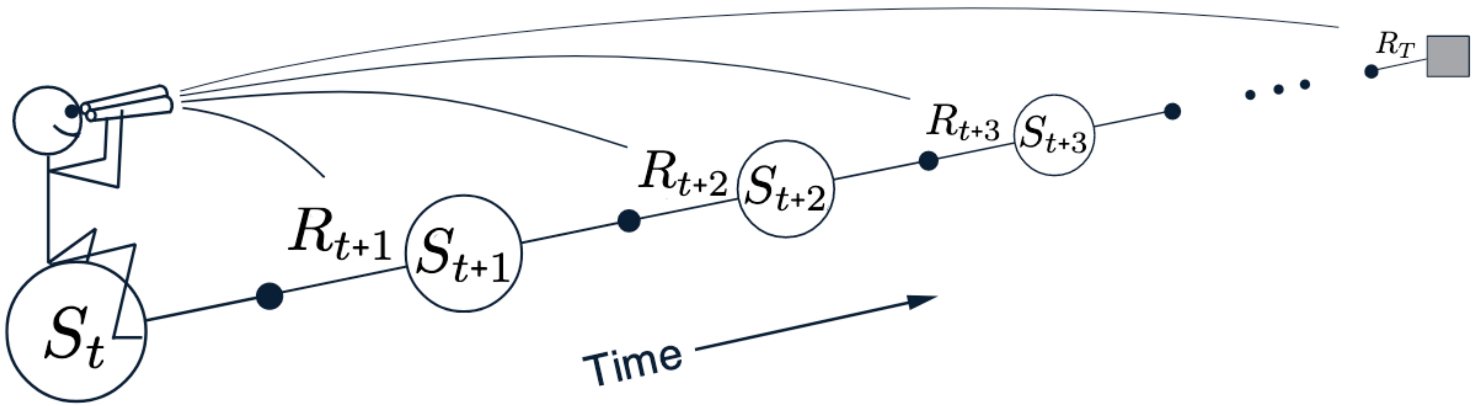
λ return

- The one-step return is given the largest weight, $1 - \lambda$; the two-step return is given the next largest weight, $(1 - \lambda)\lambda$; the three-step return is given the weight $(1 - \lambda)\lambda^2$; and so on…
- What do we get if we set $\lambda = 0$?
  - One-step return: TD(0)
- What do we get if we set $\lambda = 1$?
  - Monte Carlo: TD(1)
  - (For the bounded horizon version)



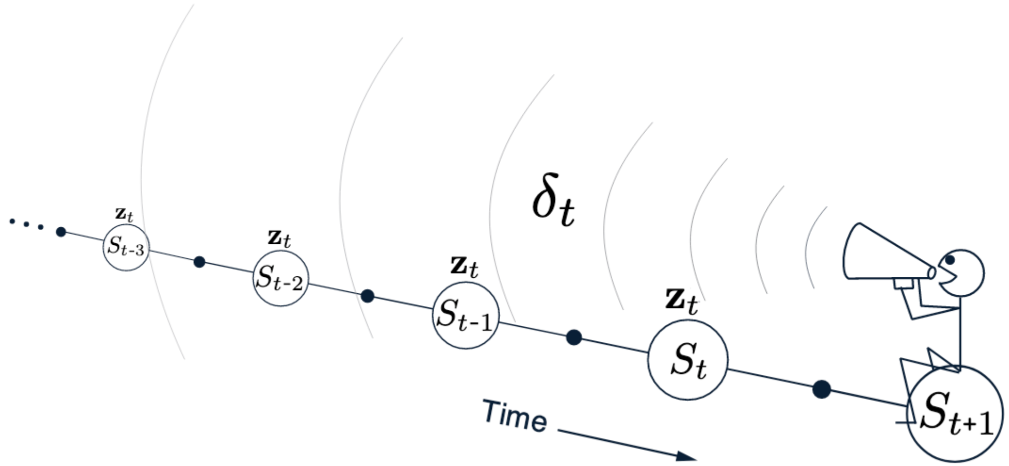
λ return

- The λ-return gives us an alternative way of moving smoothly between Monte Carlo and one-step TD methods that can be compared with the n-step TD (both estimate G_t)
  - For approximation methods: $\theta_{t+1} = \theta_t + \alpha\left[G_t^\lambda - \hat{v}(S_t;\theta_t)\right]\nabla\hat{v}(S_t;\theta_t)$
- For each visited state, we look into the future to determine the λ return



TD(λ)

- Looking into the future requires post-mortem updates
- We would like to use λ return online:
  - Update the weights on every step of an episode rather than only at the end
  - Equally distributed computation time rather that all at the end of the episode
  - Applicable to continues control problems rather than just episodic problems
- Update the past: once reached step *n* compute the impact on previous states



Tabular TD(λ) for estimating v_π

- Episode (s_0, π):
  - $Z \leftarrow [0]^{|S|}$
  - $t = 0$
  - $while\ t < horizon\ or\ s_t\ not\ finale$:
    - $A_t = \pi(S_t)$
    - Perform A_t and observe R_t, S_{t+1}
    - $Z(S_t) = Z(S_t) + (1 - \lambda)$
    - $v = v + \alpha\big(R_t + \gamma v(\mathrm{S}_{\mathrm{t+1}}) - v(\mathrm{S}_{\mathrm{t}})\big)z$
    - $Z = \boldsymbol{\lambda}\gamma Z$
    - $S_t, = S_{t+1}$

The eligibility to learn from the last action

In many implementations this will be replaced with 1: no problem as this is a constant that can be seen as part of the learning rate α (the weights won't sum to 1 though)

Tabular TD(λ) for estimating v_π

- Episode (s_0, π):
  - $Z \leftarrow [0]^{|S|}$
  - $t = 0$
  - $while\ t < horizon\ or\ s_t\ not\ finale$:
    - $A_t = \pi(S_t)$
    - Perform A_t and observe R_t, S_{t+1}
    - $Z(S_t) = Z(S_t) + (1 - \lambda)$
    - $v = v + \alpha\left(R_t + \gamma v(\mathrm{S}_{t+1}) - v(\mathrm{S}_t)\right)z$
    - $Z = \boldsymbol{\lambda}\gamma Z$
    - $S_t, = S_{t+1}$

Update all states based on their current eligibility (vector notation)

The eligibility to learn from the last action

In many implementations this will be replaced with 1: no problem as this is a constant that can be seen as part of the learning rate α (the weights won't sum to 1 though)

The TD error (δ)

The eligibility vector for this update

Tabular TD(λ) for estimating v_π

- Episode (s_0, π):
  - $Z \leftarrow [0]^{|S|}$
  - $t = 0$
  - $while\ t < horizon\ or\ s_t\ not\ finale$:
    - $A_t = \pi(S_t)$
    - Perform A_t and observe R_t, S_{t+1}
    - $Z(S_t) = Z(S_t) + (1 - \lambda)$
    - $v = v + \alpha\left(R_t + \gamma v(S_{t+1}) - v(S_t)\right) z$
    - $Z = \lambda \gamma Z$
    - $S_t, = S_{t+1}$

Update all states based on their current eligibility (vector notation)

The eligibility to learn from the last action

In many implementations this will be replaced with 1: no problem as this is a constant that can be seen as part of the learning rate α (the weights won't sum to 1 though)

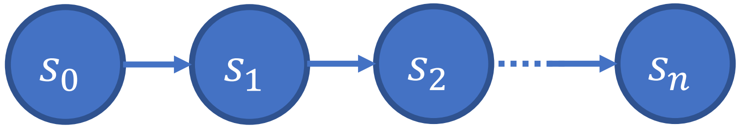
Decay the learning impact of future reward. $0 < \lambda < 1$

The TD error (δ)

The eligibility vector for this update

TD(λ)

- First time $v(S_t)$ is updated (right after S_t is visited at step t)
  - $z_t(S_t) = 1 - \lambda$
- $v(S_t)$ will continue to be updated at each future step with diminishing eligibility
  - $z_{t+1}(S_t) = \lambda\gamma z(S_t) = (1-\lambda)\lambda\gamma$
  - $z_{t+2}(S_t) = \lambda\gamma z(S_t) = (1-\lambda)\lambda^2\gamma^2$
  - … $z_{t+n}(S_t) = \lambda\gamma z(S_t) = (1-\lambda)\lambda^n\gamma^n$
- After visiting S_n , update $S_0 = S_0 + z_n(S_0) \cdot \alpha(R_{n+1}+\gamma v(S_{n+1}) - v(S_n))$



TD(λ) with function approximation

- With function approximation, the eligibility trace is a vector $z_t \in \mathbb{R}^d$ with the same number of components as the tunable parameters vector (θ)
- Whereas the tunable parameters vector is a long-term memory, accumulating over the lifetime of the system, the eligibility trace is a short-term memory, typically lasting less time than the length of an episode
- At each time step the eligibility vector is updated with the addition of the gradient of the approximation function $z_t = \lambda\gamma z_{t-1} + \nabla\hat{v}(S_t; \theta)$
- Keeps track of which tunable parameter contributed, positively or negatively, to recent state valuations, where “recent” is defined in terms of λ and γ

TD(λ) + approximation

- The trace indicates the eligibility of each component of the weight vector for undergoing learning changes should a reinforcing event occur
- The reinforcing events we are concerned with are the moment-by-moment one-step TD errors:
  - $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}; \theta) - \hat{v}(S_t; \theta)$
- The tunable parameters vector is updated on each step proportional to the scalar TD error and the vector of eligibility traces:
  - $\theta_{t+1} = \theta_t + \alpha \delta_t z_t$

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

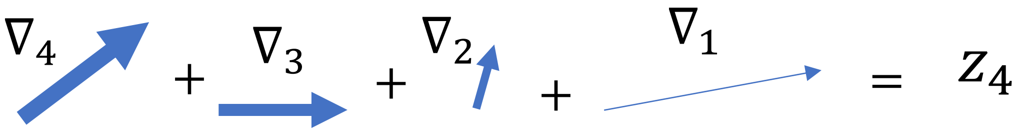
Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

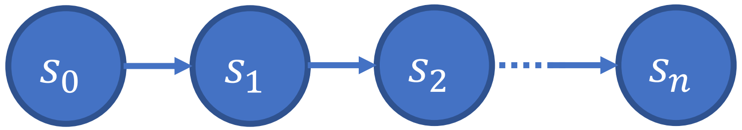
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
    Initialize S
    $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)
    Repeat (for each step of episode):
    . Choose $A \sim \pi(\cdot|S)$
    . Take action A, observe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

Eligibility for θ_i is determined by the partial derivative for θ_i (can be positive or negative) and decays exponentially

Semi-gradient TD(λ)

- First update based on $\nabla\hat{v}(S_t)$ (right after S_t is visited at step t)
  - $z_t = \nabla\hat{v}(S_t;\theta)$
  - $\theta = \theta + \alpha\delta z$
- Next updates based on $\nabla\hat{v}(S_t)$ will have diminishing eligibility
  - $z_{t+1} = \lambda\gamma z_t + \nabla\hat{v}(S_{t+1};\theta) = \lambda\gamma\nabla\hat{v}(S_t;\theta) + \nabla\hat{v}(S_{t+1};\theta)$
  - $z_{t+2} = \lambda^2\gamma^2\nabla\hat{v}(S_t;\theta) + \lambda\gamma\nabla\hat{v}(S_{t+1};\theta) + \nabla\hat{v}(S_{t+2};\theta)$
  - … $z_{t+n} = \lambda^n\gamma^n\nabla\hat{v}(S_t;\theta) + \lambda^{n-1}\gamma^{n-1}\nabla\hat{v}(S_{t+1};\theta) + \cdots + \lambda\gamma\nabla\hat{v}(S_{t+n-1};\theta) + \nabla\hat{v}(S_{t+n};\theta)$
- The eligibility of the gradient decays exponentially





Theoretical properties

- Linear approximation $TD(\lambda)$ is proven to converge in the on-policy case if the step-size parameter is reduced over time according to the usual conditions (slide 18 in 2.Multi-armed_bandits)
- Convergence is not to the minimum-error weight vector, but to a nearby weight vector that depends on λ
- The error is bounded:
  - $\overline{VE}(\theta_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma} \min_\theta \overline{VE}(\theta)$
- That is, the asymptotic error is no more than $\frac{1-\gamma\lambda}{1-\gamma}$ times the smallest possible error under the current approximation
- As λ approaches 1, the bound approaches the minimum error (full episode return is unbiased), and it is loosest at λ= 0
- In practice, $\lambda = 1$ is often the poorest choice due to training instabilities (high var return)

Learning action values

- Very few changes are required in order to extend eligibility-traces to action-value methods
- Learn approximate action values, $\hat{Q}(s, a; \theta)$, rather than approximate state values, $\hat{V}(s; \theta)$
  - Use the action-value form of the n-step return
  - $G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}; \theta_{t+n-1}), \quad t + n < T$
  - $G_{t:t+n}^{\lambda} = (1 - \lambda) \sum_{k=1}^{n-1} \lambda^{k-1} G_{t:t+k} + \lambda^{n-1} G_{t:t+n}$
- For control problems learn $\hat{Q}$ instead of $\hat{V}$
  - $\theta_{t+1} = \theta_t + \alpha \left[G_t^{\lambda} - \hat{q}(S_t, A_t; \theta) \right] \nabla \hat{Q}(S_t, A_t; \theta), \quad t = 0, \ldots, T - 1$
- That's it, TD(λ) will now converge* on action values

Sarsa(λ)

- The TD(λ) action value variant is known as $\text{SARSA}(\lambda)$
  - $z_{-1} = [0]$
  - $z_t = \gamma \lambda z_{t-1} + \nabla \hat{Q}(S_t, A_t; \theta_t)$
  - $\delta_t = R_{t+1} + \gamma \hat{Q}(S_{t+1}, A_{t+1}; \theta_t) - \hat{Q}(S_t, A_t; \theta_t)$
  - $\theta_{t+1} = \theta_t + \alpha \delta_t z_t$

Online SARSA with Linear approximator

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

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ s.t. $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: the policy π to be evaluated, if any

Initialize parameter $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Loop for each episode:
    Initialize S
    Choose $A \sim \pi(\cdot|S)$ or near greedily from S using $\mathbf{w}$
    $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
    $\mathbf{z} \leftarrow \mathbf{0}$
    $Q_{old} \leftarrow 0$ (a scalar temporary variable)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose $A' \sim \pi(\cdot|S')$ or near greedily from S' using $\mathbf{w}$
        $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
        $Q \leftarrow \mathbf{w}^\top\mathbf{x}$
        $Q' \leftarrow \mathbf{w}^\top\mathbf{x}'$
        $\delta \leftarrow R + \gamma Q' - Q$
        $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\boxed{\mathbf{x}}$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\boxed{\mathbf{x}}$
        $Q_{old} \leftarrow Q'$
        $\mathbf{x} \leftarrow \mathbf{x}'$
        $A \leftarrow A'$
    until S' is terminal

Q learning with eligibility traces

- Off policy Q learning attempts to learn Q^* which is unknown
- In order to avoid convergence to local optimum, Q learning must act randomly e.g., through an epsilon greedy approach
- Eligibility traces assume that the underlying trajectory is sampled from the evaluated policy (on policy)
- How should we update eligibilities for Q learning once a non-greedy action is taken?

Watkins's $Q(\lambda)$

- Zero out eligibility trace after a non-greedy action
- $z_t(s,a) = \begin{cases} 1 + \gamma\lambda z_{t-1}(s,a) & S = S_t, A = A_t, Q_{t-1}(S_t, A_t) = \max_{\text{a}} Q_{t-1}(S_t, a) \\ 0 & Q_{t-1}(S_t, A_t) \neq \max_{\text{a}} Q_{t-1}(S_t, a) \\ \gamma\lambda z_{t-1}(s,a) & else\ (s, a \text{ was not visitied at } t \text{ and greedy action}) \end{cases}$
- $\delta_t = R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q_t\,(S_t, A_t)$
- $Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t z_t(s,a)$

Watkins's $Q(\lambda)$ – tabular version

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a

Repeat (for each episode) :

  Initialize s, a

  Repeat (for each step of episode) :

    Take action a, observe r, s'

    Choose a' from s' using policy derived from Q (e.g. ? - greedy)

    $a^* \leftarrow \arg\max_b Q(s', b)$ (if a ties for the max, then $a^* \leftarrow a'$)

    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a^*)$

    $e(s,a) \leftarrow e(s,a) + 1$

    For all s,a :

      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$

      If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$

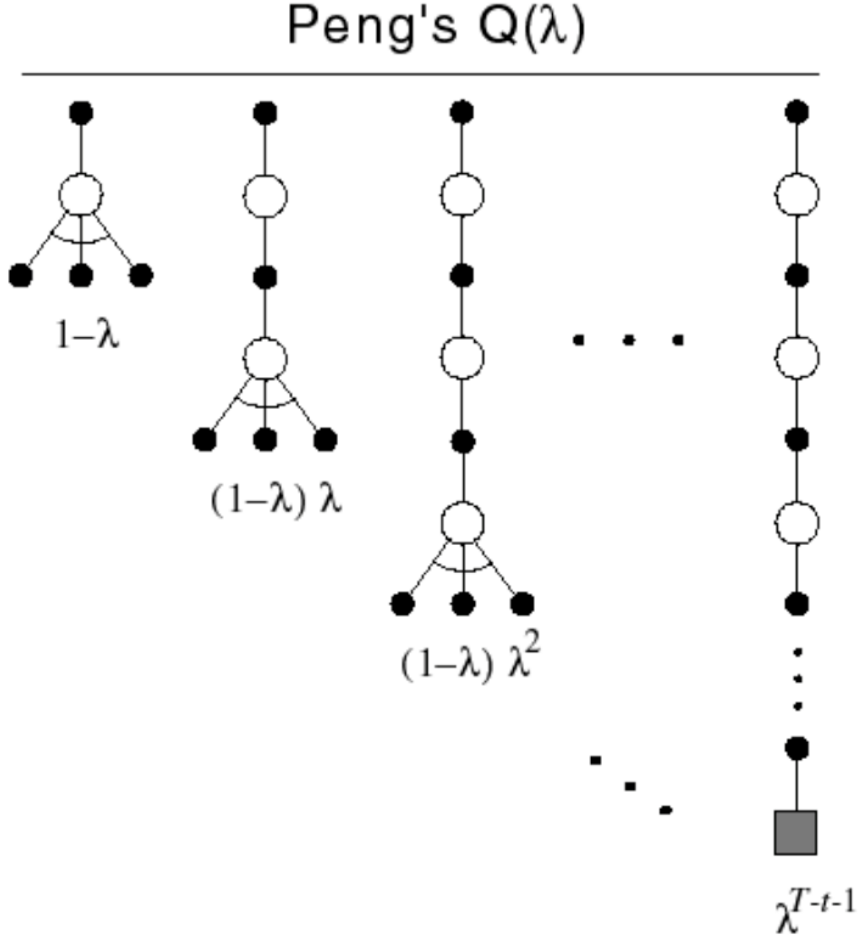
        else $e(s,a) \leftarrow 0$

    $s \leftarrow s'; a \leftarrow a'$

  Until s is terminal

Peng's $Q(\lambda)$

- Disadvantage of Watkins's method
  - The eligibility trace will be "cut" (zeroed out) frequently resulting in little advantage to traces
- Peng:
  - Backup max action except at end
  - Never cut traces
  - Disadvantage: Complicated to implement



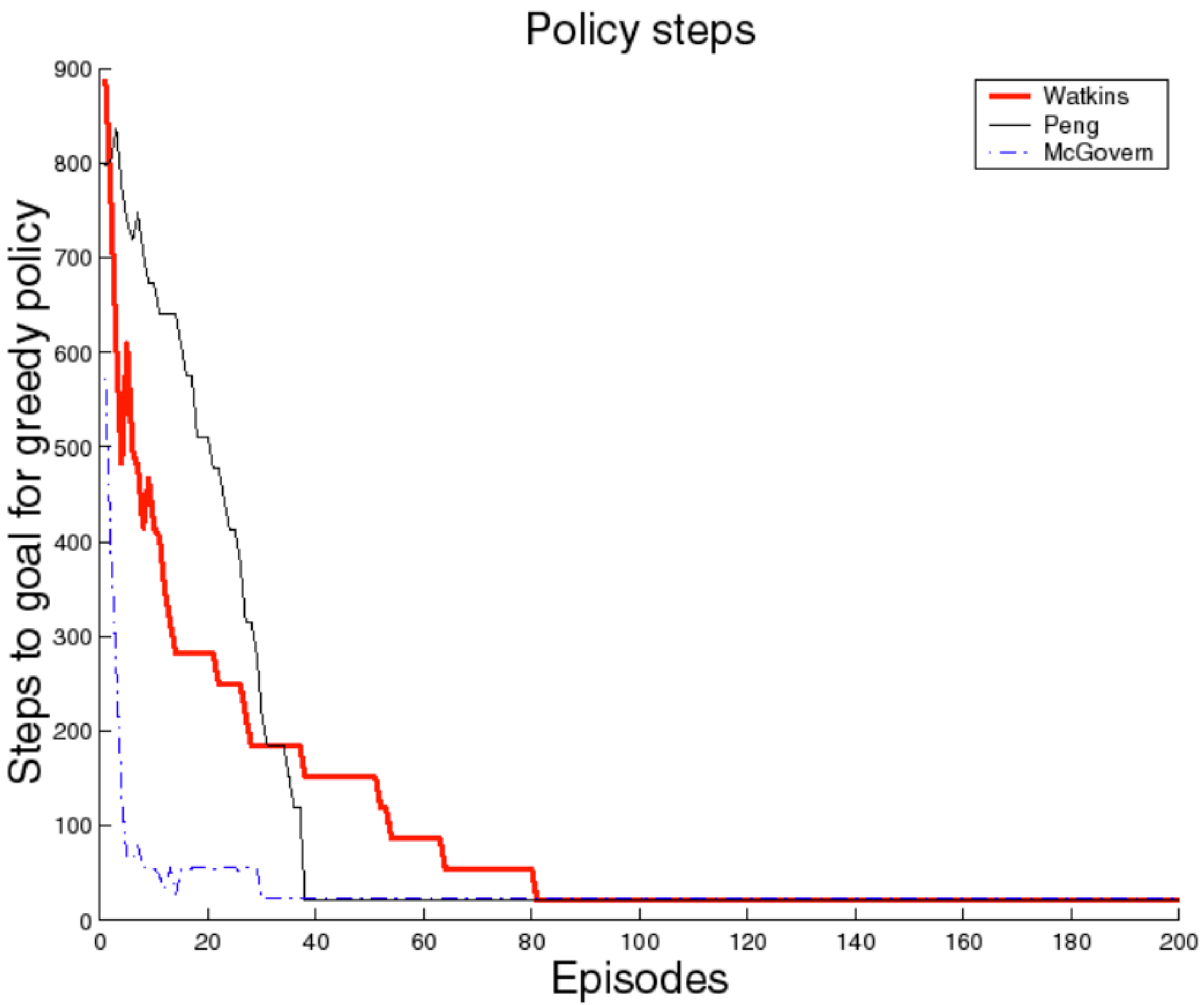
Naïve $Q(\lambda)$

- Never zero traces
- Always backup values at current action (even if not maximizing q value)
- $z_t(s,a) = \begin{cases} 1 + \gamma\lambda z_{t-1}(s,a) & S = S_t \\ \gamma\lambda z_{t-1}(s,a) & else \end{cases}$

Comparing the three approaches

- Compared Watkins's, Peng's, and Naïve (called McGovern's here) $Q(\lambda)$
- Deterministic grid world with obstacles
  - 10x10 gridworld
  - 25 randomly generated obstacles
  - 30 runs
  - $\alpha = 0.05, \ \gamma = 0.9, \ \lambda = 0.9, \ \varepsilon = 0.05$, accumulating traces

*See McGovern and Sutton (1997). Towards a Better $Q(\lambda)$ for other tasks and results (stochastic tasks, continuing tasks, etc)



From McGovern and Sutton (1997). Towards a better Q(λ)

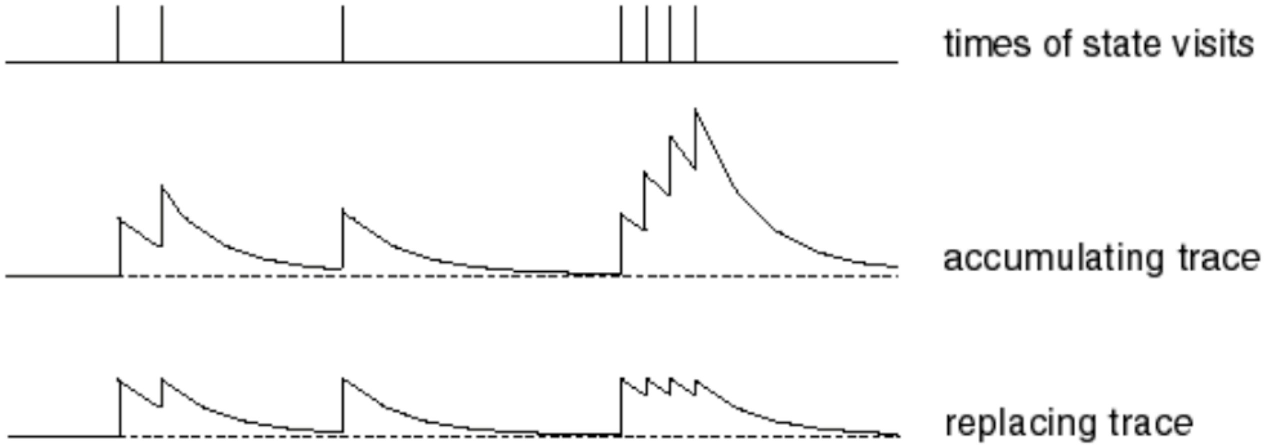
Convergence for $Q(\lambda)$

- None of the methods are proven to converge
- Watkins's is thought to converge to Q*
- Peng's is thought to converge to a mixture of Q_π and Q^*

Replacing traces

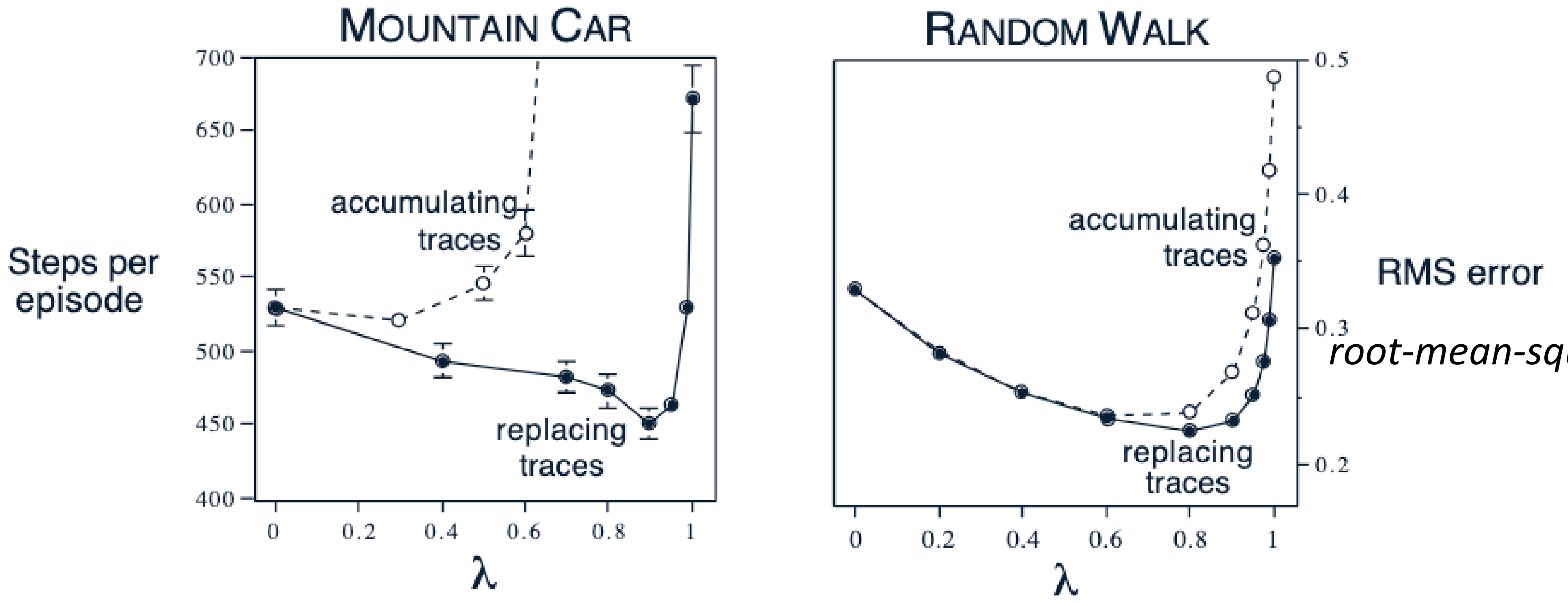
- Using accumulating traces, frequently visited states can have eligibilities greater than 1
- This can be a problem for convergence
- Replacing traces: Instead of adding 1 when you visit a state, set that trace to 1

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases}$$



Replacing traces

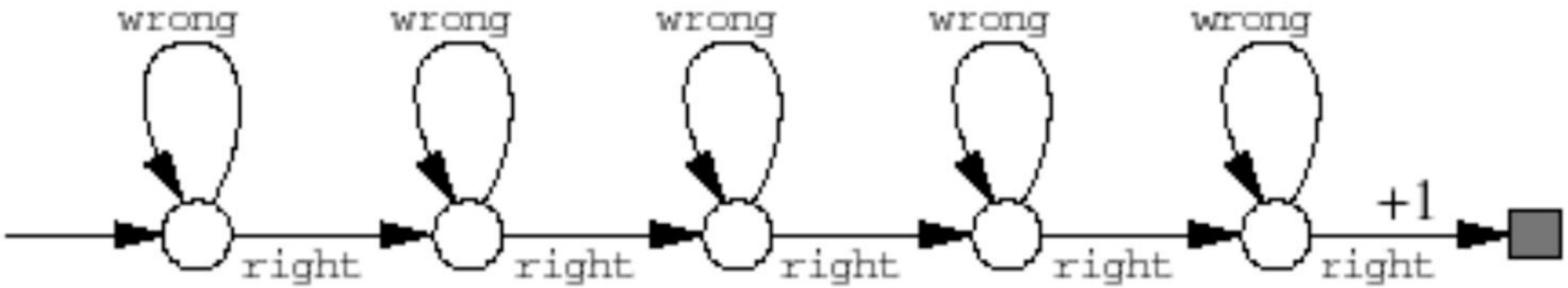
- Replacing traces perform better than accumulating traces over more values of λ



root-mean-square

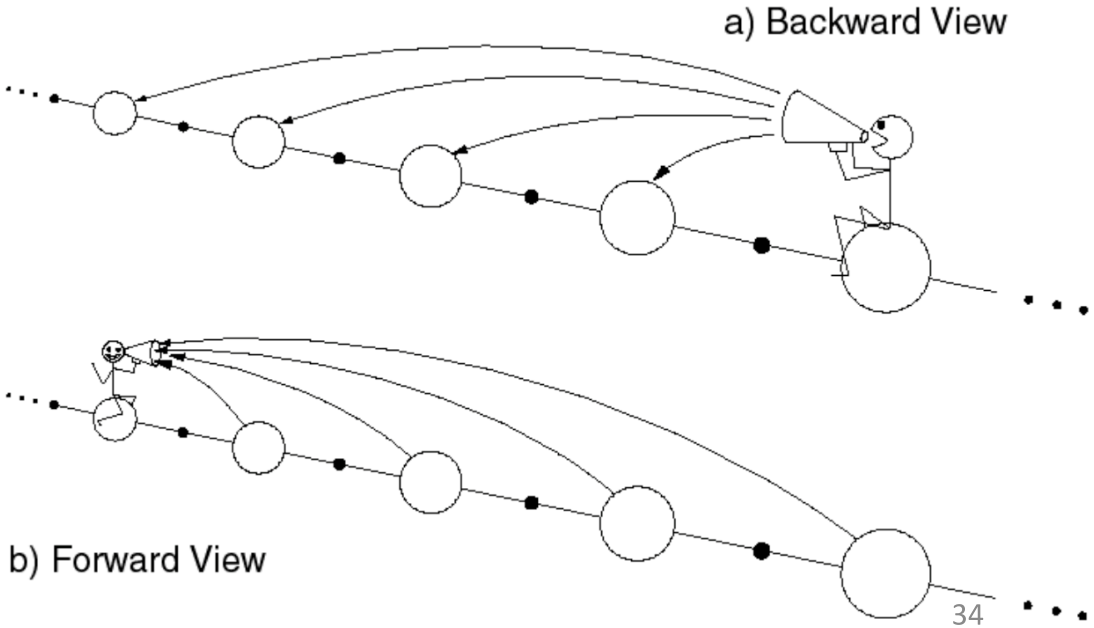
Why Replacing Traces?

- What will happen in the following domain if accumulated traces is used?
- Will the learning be meaningful?



What did we learn?

- $TD(\lambda)$ Provides efficient, incremental way to combine MC and TD(0)
- Advantages of MC (lower bias)
- Advantages of TD (low variance)
- Can significantly speed up learning
- Does have a cost in computation
- Results in on policy learning
- Can extend to off policy
  - Watkins's $Q(\lambda)$
  - At the cost of less efficient learning



What next?

- **Lecture**: Deep Q-Learning
- **Assignments**:
  - Q-Learning with Approximation, Oct. 14, EOD
- **Quiz (on Canvas)**:
  - Eligibility traces, Oct 21, EoD
- **Project**:
  - Literature survey, by Monday, November 4 EOD