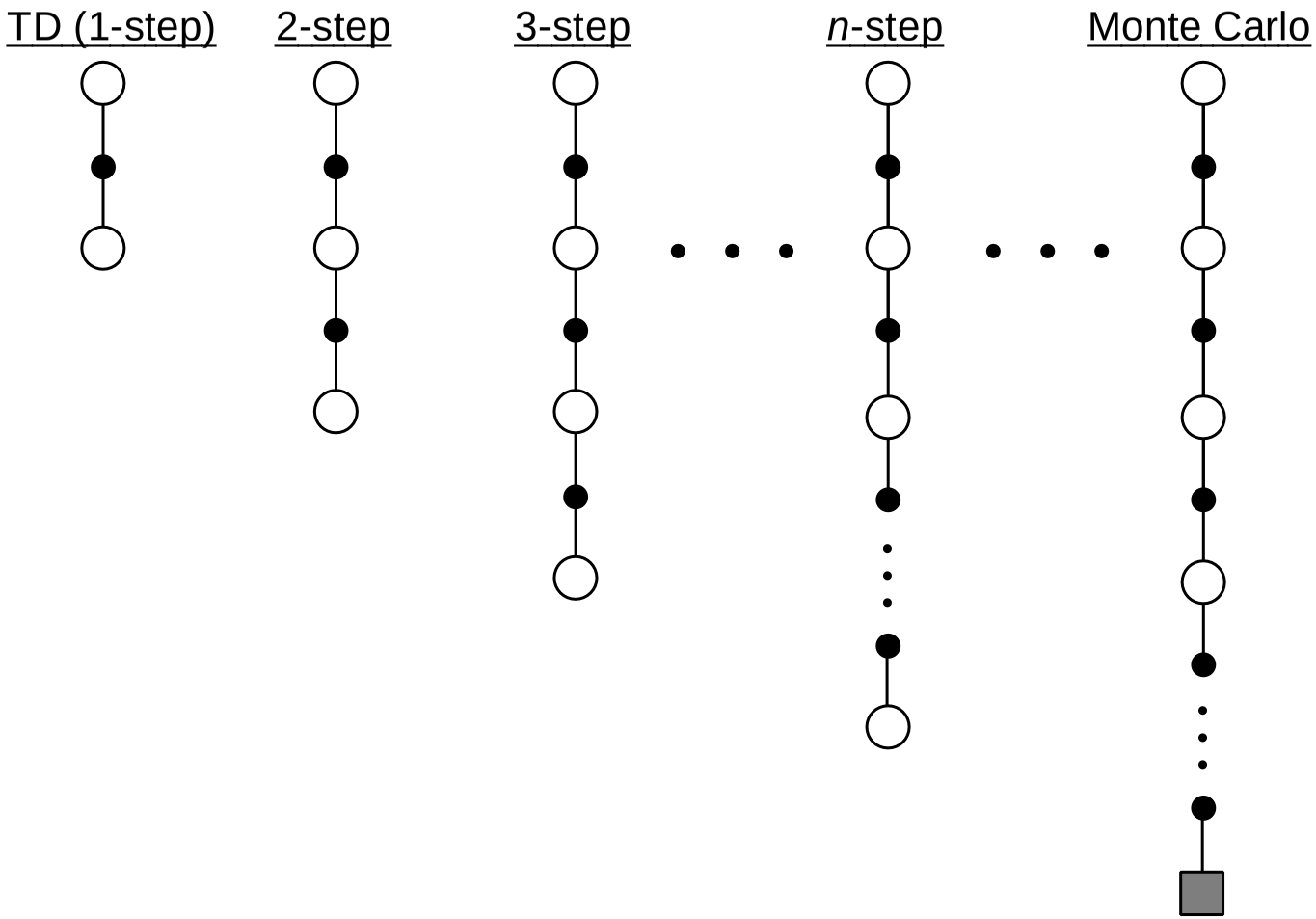
Lecture 20: Eligibility Traces

- N-step predictions
- Eligibility traces: $TD(\lambda)$
- Eligibility traces for control algorithms
- Replacing vs. accumulating traces

N-step TD predictions

Main idea: look farther into the future when you do a TD backup



Mathematics of N-step TD prediction

- Monte Carlo algorithms use the full return as a target:

$$R_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-t-1} r_T$$

- TD uses the approximate value function $\hat{V}$ to estimate the return after the first step:

$$R_t^{(1)} = r_{t+1} + \gamma \hat{V}(s_{t+1})$$

- But we could truncate the return after any number of steps, and use $\hat{V}$ as an approximation for the rest. E.g. 2-step return:

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 \hat{V}(s_{t+2})$$

- In general, an *N-step return* is:

$$R_t^{(N)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{N-1} r_{t+N} + \gamma^N \hat{V}(s_{t+N})$$

Learning with N-step backups

- Update rule can be done on-line or off-line:

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha \left[R_t^{(N)} - \hat{V}(s_t) \right]$$

- N-step returns also have a contraction property, just like TD:

$$\max_s |E_\pi \left\{ R_t^{(N)} \mid s_t = s \right\} - V^\pi(s)| \leq \gamma^N \max_s |\hat{V}(s) - V^\pi(s)|$$

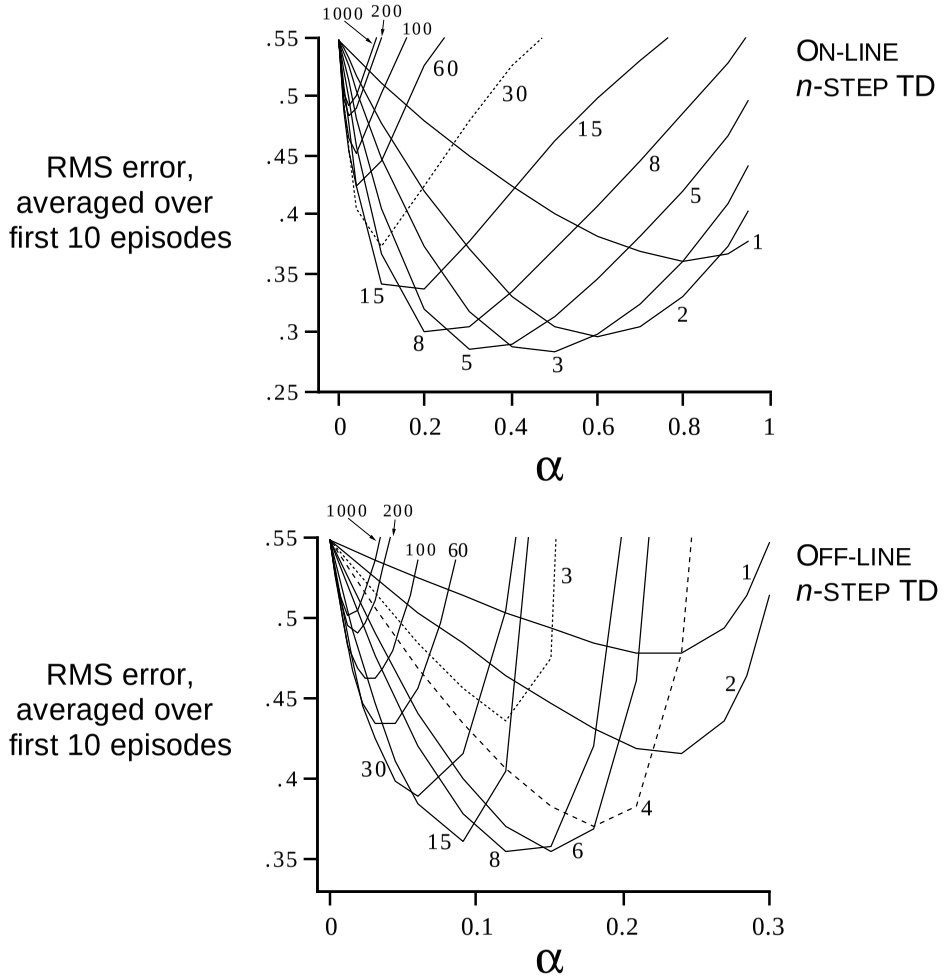
  In other words, maximum error when using the N-step return is less than maximum error using $\hat{V}$

- Using this, we can show that the N-step learning method converges for any N

Issue: what is a good N?

Some empirical evidence

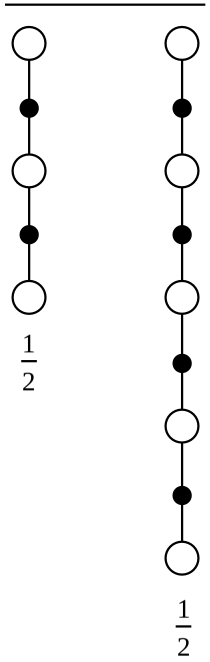
Random walk task, 19 states



Averaging N-step returns

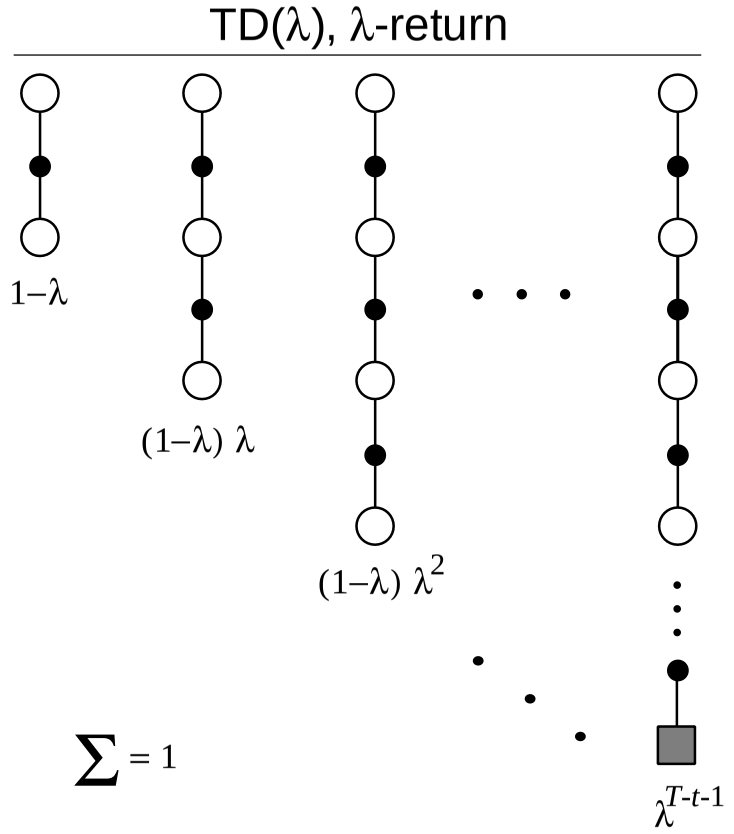
- Idea: backup an average of N-step returns

  E.g. Half of a 2-step and half of a 4-step return



- Since any N-step return reduces the error, any linear combination of them will also reduce the error
- $TD(\lambda)$ is a particular way of averaging *all* the N-step backups

Forward view of $TD(\lambda)$



λ-return:

$$R_t^\lambda = (1-\lambda)\sum_{N=1}^{\infty} \lambda^{N-1} R_t^N,$$

where λ is a parameter between 0 and 1

Backup using λ-return:

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha\left[R_t^\lambda - \hat{V}(s_t)\right]$$

This weighs each N-step backup by a weight λ^{N-1}, depending on the time since the state was visited.

Relationship to TD and Monte Carlo

Suppose we have a trial-base task, and the trial ends at T. The λ-return can be re-written as:

$$R_t^\lambda = (1-\lambda)\sum_{N=1}^{T-t-1}\lambda^{N-1}R_t^N + \lambda^{T-t-1}R_t$$

The first term shows truncated returns until termination, the last one shows the whole return for the trial

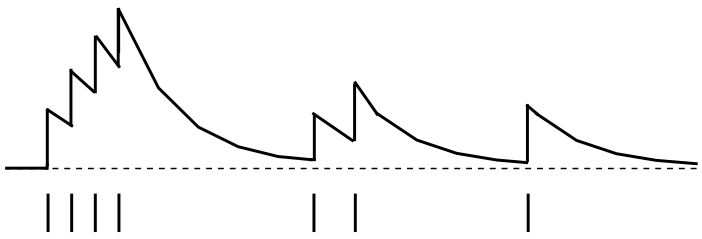
- If $\lambda = 1$, we get $R_t^\lambda = R_t$ - Monte Carlo!
- If $\lambda = 0$, we get $R_t^\lambda = R_t^{(1)}$ - Temporal difference! (we will call it TD(0) from now on)

λ provides a way of interpolating between TD(0) and Monte Carlo!

Eligibility traces

- The previous algorithm is helpful for understanding but is not convenient to implement directly
- For a convenient implementation, we use an extra variable for each state, $e_t(s)$, called the *eligibility trace*, which keeps track of how long ago the state was visited
- Update rule for the *accumulating eligibility trace*:

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



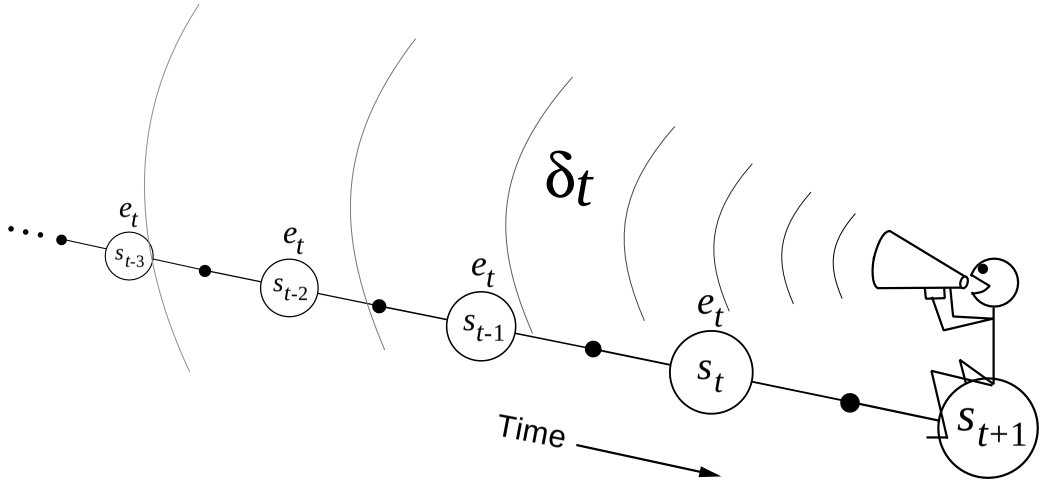
On-line tabular $TD(\lambda)$

1. Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all states s
2. Pick a start state s
3. Repeat for every time step:
   (a) Choose action a based on policy π and the current state s
   (b) Take action a, observe immediate reward r and new state s'
   (c) Compute the TD error: $\delta \leftarrow r + \gamma V(s') - V(s)$
   (d) Mark the current state as visited: $e(s) \leftarrow e(s) + 1$
   (e) For all states s, update the value function and eligibility trace:

$$\begin{aligned} V(s) &\leftarrow V(s) + \alpha\delta e(s) \\ e(s) &\leftarrow \gamma\lambda e(s) \end{aligned}$$

   (f) $s \leftarrow s'$

What $TD(\lambda)$ does



- On every time step t, we compute the TD error:

$$\delta_t = r_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$$

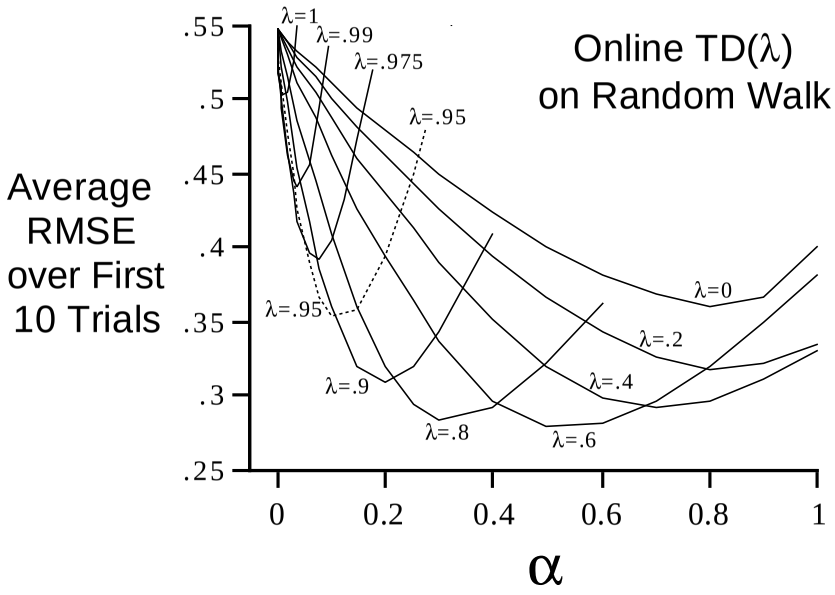
- Shout δ_t backwards to past states
- The strength of your voice decreases with temporal distance by $\gamma\lambda$

Relation of the on-line algorithm to TD(0) and Monte Carlo

- We can show (by algebraic manipulations) that the two algorithms discussed so far are equivalent
- In the on-line algorithm, setting $\lambda = 0$ gives us TD(0) (just like before)
- Setting $\lambda = 1$ gives us Monte Carlo, also as before - we call this TD(1)

  But this is a *better implementation of Monte Carlo!*
  - Straightforward to apply to continuing tasks
  - Works incrementally and on-line, instead of waiting until the end of the episode

Typical empirical results

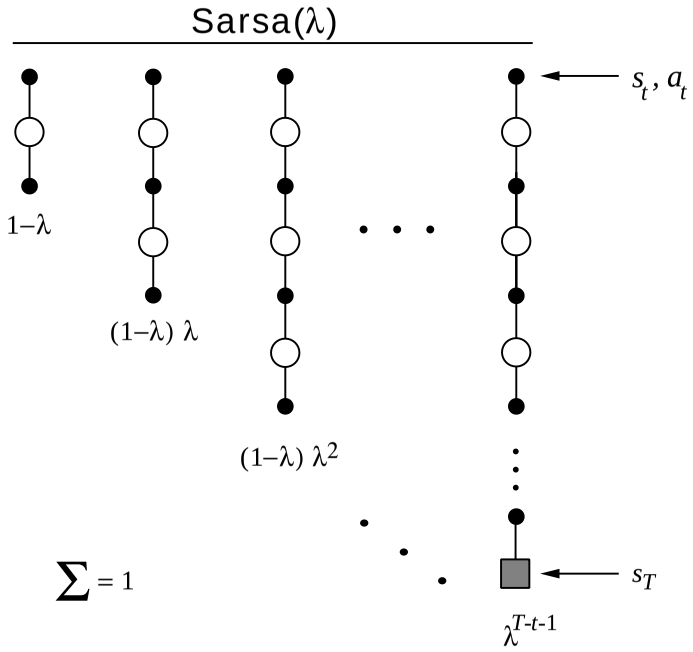


- Typically intermediate values of λ converge the fastest
- A sharp decrease in performance happens at λ very close to 1
- The algorithm converges in the limit, with probability 1, to correct values, for any λ

Eligibility traces for control: Sarsa(λ)

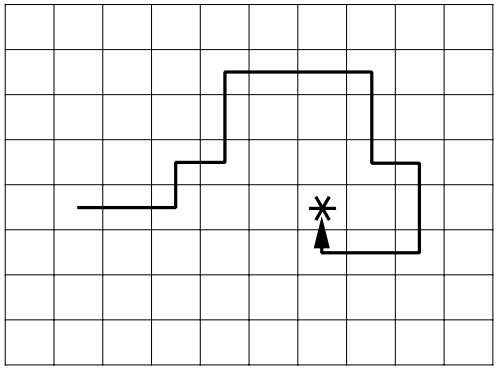
Just like $TD(\lambda)$, except there is an eligibility trace for every state-action pair:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t \\ \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t \end{cases}$$

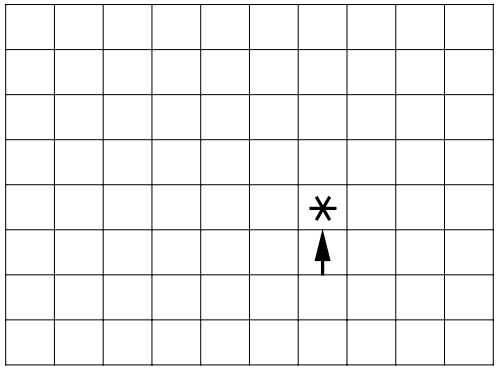


Gridworld example

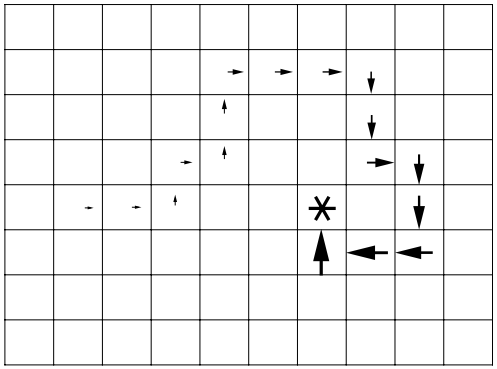
Path taken



Action values increased by one-step Sarsa



Action values increased by Sarsa(λ) with λ=0.9



- Even from just one trial, the agent has much more information about getting to the goal
- More state-action pairs get values, even though their policy is not correct at this point
- Learning can become considerably faster

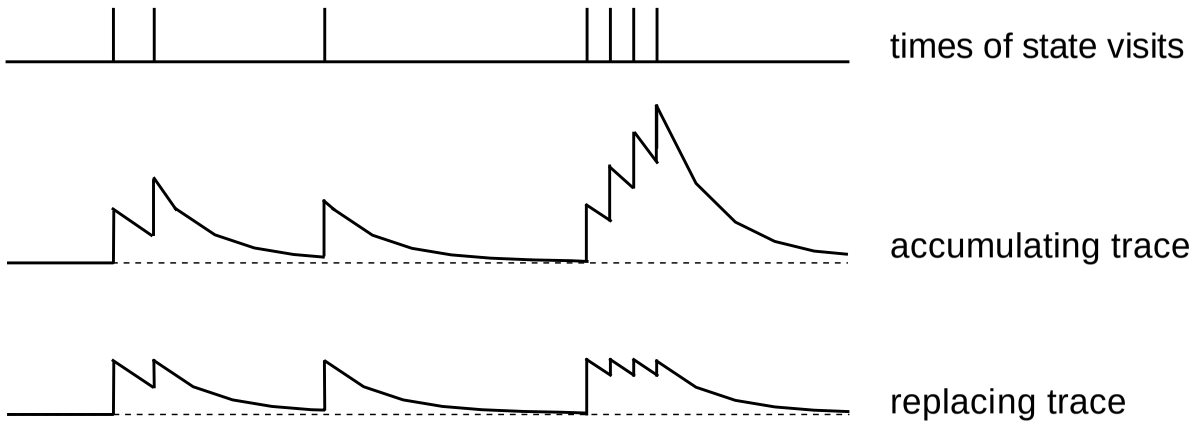
Eligibility traces for Q-learning

- Naive version: just like the Sarsa eligibility trace update, but with the error as computed by Q-learning
- But that means that backups will be done over exploratory actions as well
- Watkins: set the eligibility trace to 0 for all state-action pairs whenever an exploratory action is taken
- Watkins Q(λ) is believed to converge to Q^*, but no convergence proof exists for *any* control algorithm with eligibility traces

Replacing traces

- When using accumulating traces, frequently visited states can have eligibilities greater than 1 - problem for convergence
- *Replacing traces:* instead of incrementing the trace by 1 when a state is visited, *set the trace to 1*:

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



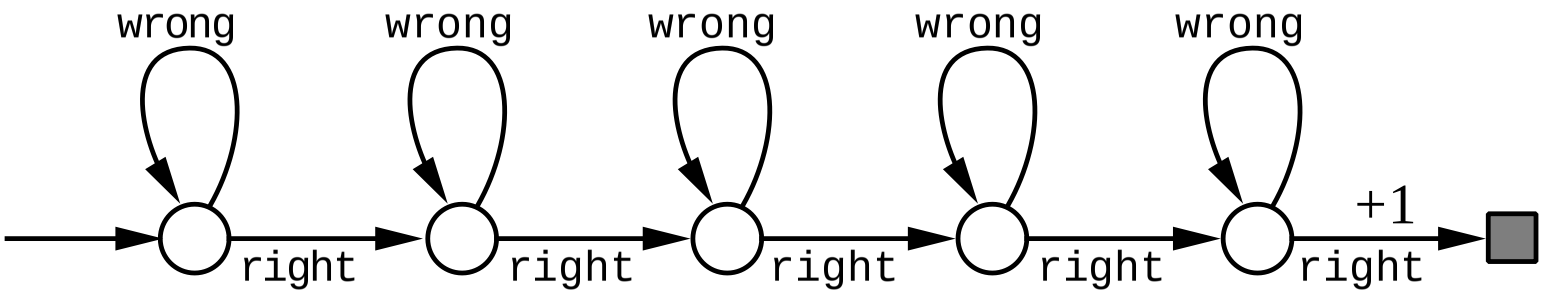
Replacing traces for action values

When a state is revisited and a different action is taken, the trace for the previous action is set to 0:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \text{ and } a = a_t \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t \end{cases}$$

Why replacing traces?

- Replacing traces can significantly speed learning
- Good performance over a broader range of parameters
- Accumulating cases can do especially poorly for certain kinds of tasks:



- In the undiscounted case, for $\lambda = 1$:
  - Accumulating traces are equivalent to every-visit Monte Carlo
  - Replacing traces are equivalent to first-visit Monte Carlo

Implementation issues

- A naive implementation would update all states (or state-action pairs) on every time step
- But in practice, for most values of γ and λ, the eligibility traces are very near zero for all states except those most recently visited
- A clever implementation can keep track only of the states with non-zero traces, which makes the algorithm a few times more expensive than TD(0)
- When using function approximation, extra expense is even less (see next time).

Summary

- Eligibility traces provide an efficient, incremental way to combine temporal difference and Monte Carlo methods
- Like Monte Carlo methods, they are robust to lack of Markov property (see, e.g. Loch and Singh, 1998)
- But preserve the advantage of TD in terms of bootstrapping, incremental computation
- Can significantly speed up learning (many experiments indicate intermediate λ is consistently the best)
- But there is a cost in computation (now more than one state is updated on every step)