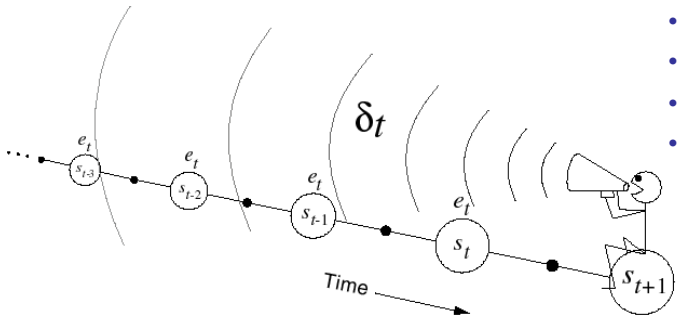
Eligibility Traces

Suggested reading:

Chapter 7 in R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction
MIT Press, 1998.

Eligibility Traces



Contents:

- n-step TD Prediction
- The forward view of TD(λ)
- The backward view of TD(λ)
- Sarsa(λ)
- Q(λ)
- Actor-critic methods
- Replacing traces

Eligibility traces

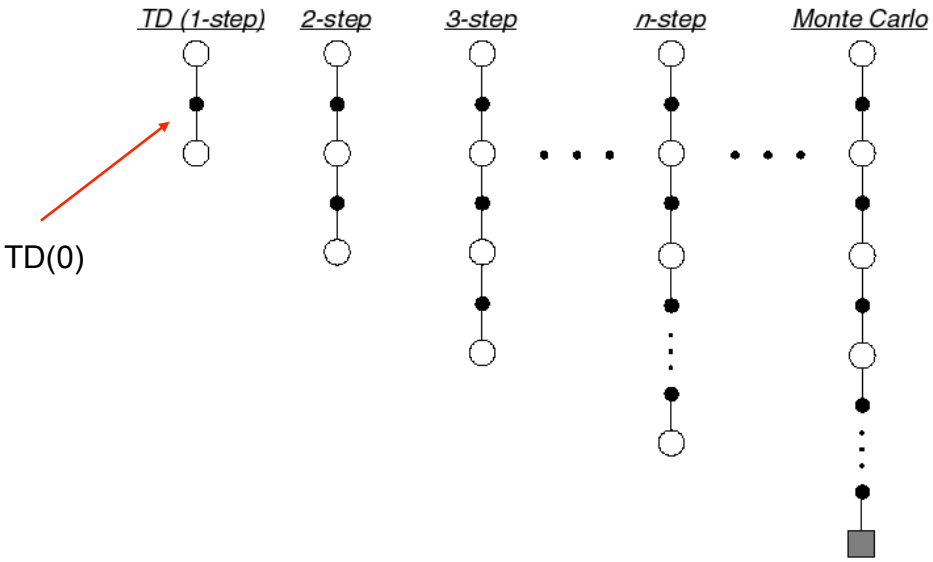
Eligibility traces are one of the basic mechanisms of reinforcement learning.

There are two ways to view eligibility traces:

- The more theoretical view is that they are a bridge from TD to Monte Carlo methods (*forward* view).
- According to the other view, an eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action (backward view). The trace marks the memory parameters associated with the event as eligible for undergoing learning changes.

n-step TD Prediction

- Idea: Look farther into the future when you do TD backup (1, 2, 3, …, n steps)



Mathematics of n-step TD Prediction

- Monte Carlo: $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T$
- TD: $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$
  - Use V to estimate remaining return
- n-step TD:
  - 2 step return: $R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$
  - n-step return: $R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$

If the episode ends in less than n steps, then the truncation in a n-step return occurs at the episode's end, resulting in the conventional complete return:

$$T - t \le n \Rightarrow R_t^{(n)} = R_t^{(T-t)} = R_t$$

n-step Backups

- Backup (on-line or off-line): $\Delta V_t(s_t) = \alpha\left[R_t^{(n)} - V_t(s_t)\right]$
- on-line: the updates are done during the episode, as soon as the increment is computed.

$$V_{t+1}(s) := V_t(s) + \Delta V_t(s)$$

- off-line: the increments are accumulated "on the side" and are not used to change value estimates until the end of the episode.

$$V(s) := V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$$

Error reduction property of n-step returns

For any V, the expected value of the n-step return using V is guaranteed to be a better estimate of V^π than V is: the worst error under the new estimate is guaranteed to be less than or equal to γ^n times the worst error under V.

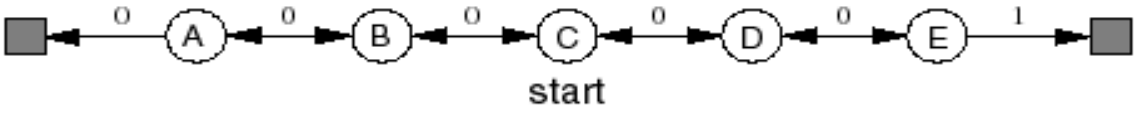
$$\max_s \left| E_\pi\{R_t^{(n)} \mid s_t = s\} - V^\pi(s) \right| \le \gamma^n \max_s \left| V(s) - V^\pi(s) \right|$$

Maximum error using n-step return — Maximum error using V

Using this, you can show that TD prediction methods using n-step backups converge.

n-step TD Prediction

Random Walk Examples:



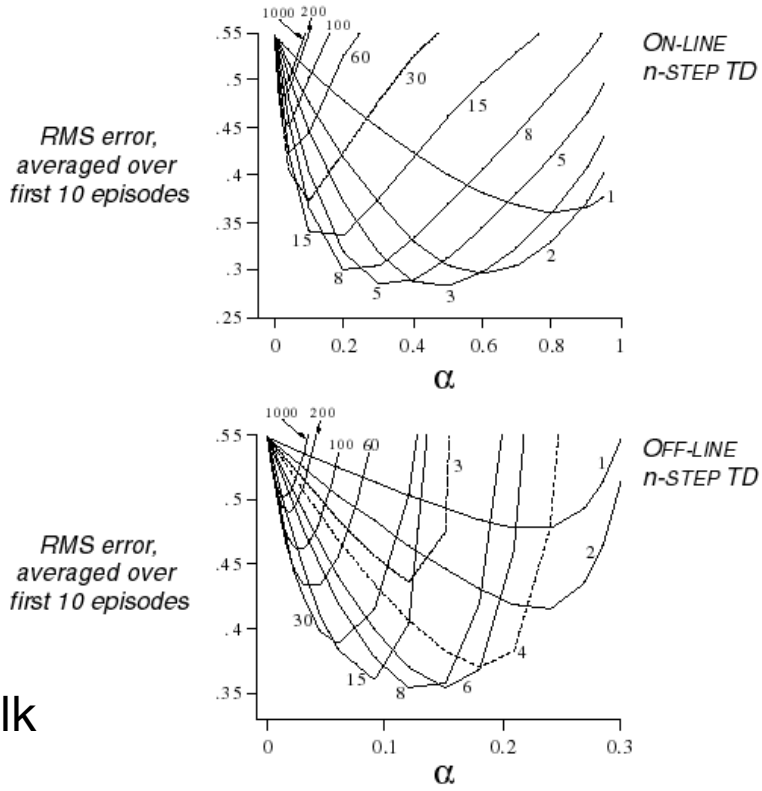
A one-step method would change only the estimate for the last state, V(E), which would be incremented toward 1, the observed return.

- How does 2-step TD work here?
- How about 3-step TD?

A two-step method would increment the values of the two states preceding termination: V(E), and V(D).

n-step TD Prediction

A larger example:



Task:

19 state random walk

Averaging n-step Returns

Backups can be done not just toward any n-step return but toward any *average of* n-step returns.

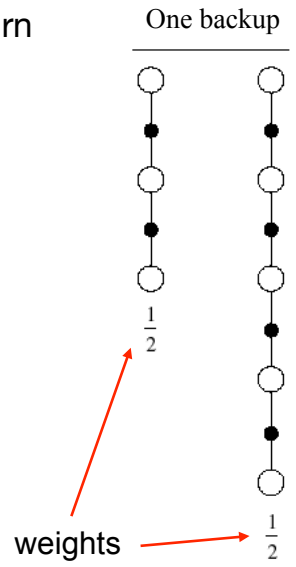
- e.g. backup half of 2-step and half of 4-step

$$R_t^{avg} = \frac{1}{2} R_t^{(2)} + \frac{1}{2} R_t^{(4)}$$

Called a complex backup

- Draw each component with a horizontal line above them
- Label with the weights for that component

positive and sum to 1



λ-return algorithm (forward view of TD(λ))

TD(λ) is a particular method for averaging all *n*-step backups.

weight by λ^{n-1} (time since visitation)

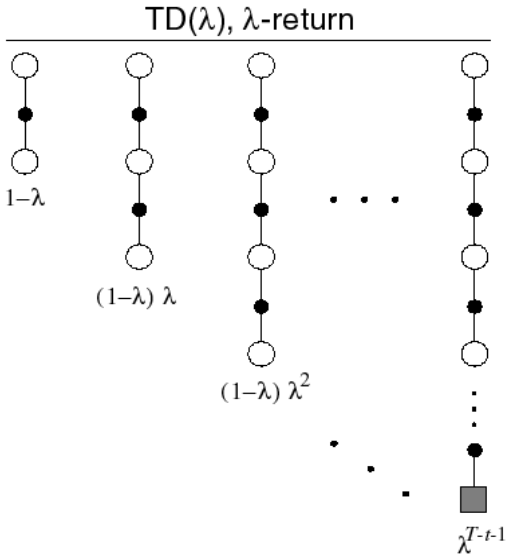
λ-return:

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

$$0 \le \lambda \le 1$$

normalization factor

$$(1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} = 1$$



Backup using λ-return:

$$\Delta V_t(s_t) = \alpha\left[R_t^\lambda - V_t(s_t)\right]$$

$$\Delta V_t(s\ ) = 0, \quad s \neq s_t$$

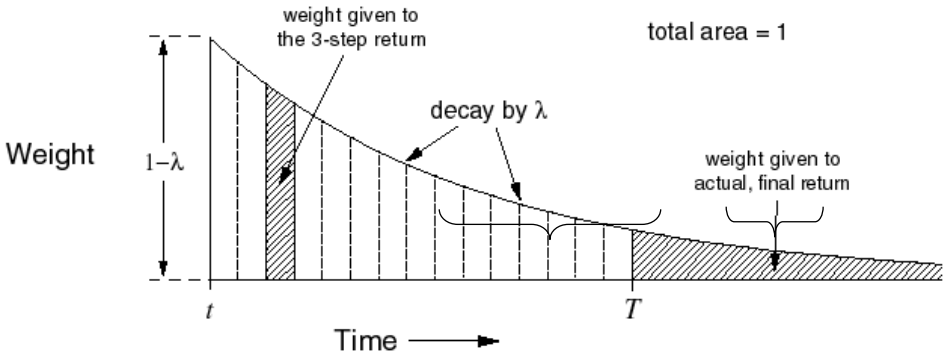
λ-return weighting Function

After a terminal state has been reached, all subsequent -step returns are equal to R_t.

$$T - t \le n \Rightarrow R_t^{(n)} = R_t^{(T-t)} = R_t$$

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

Until termination After termination



Relation to TD(0) and MC

- λ-return can be rewritten as:

$$R_t^\lambda = \underbrace{(1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}R_t^{(n)}}_{\text{Until termination}} + \underbrace{\lambda^{T-t-1}R_t}_{\text{After termination}}$$

- If λ = 1, you get MC:

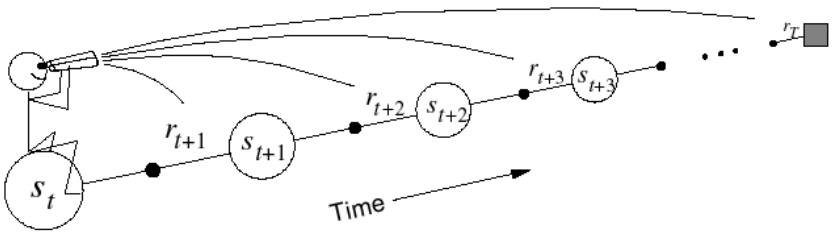
$$R_t^\lambda = (1-1)\sum_{n=1}^{T-t-1}1^{n-1}R_t^{(n)} + 1^{T-t-1}R_t = R_t$$

- If λ = 0, you get TD(0)

$$R_t^\lambda = (1-0)\sum_{n=1}^{T-t-1}0^{n-1}R_t^{(n)} + 0^{T-t-1}R_t = R_t^{(1)}$$

Forward view of TD(λ)

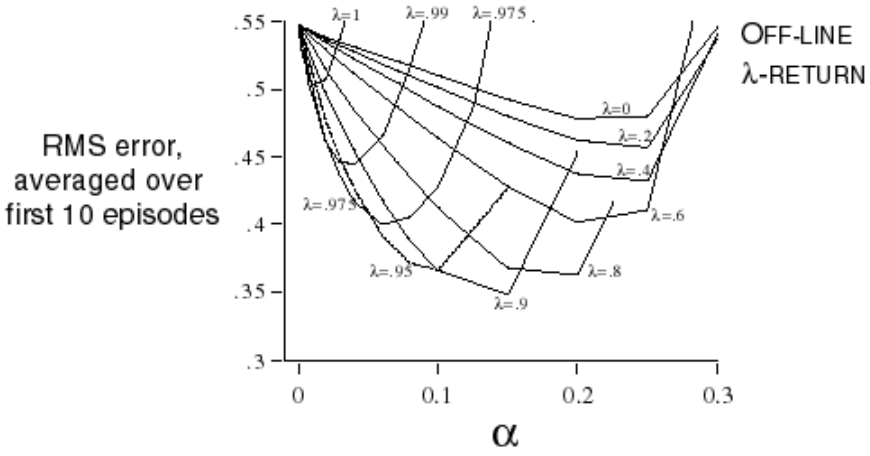
For each state visited, we look forward in time to all the future rewards and states to determine its update.



Off-line:

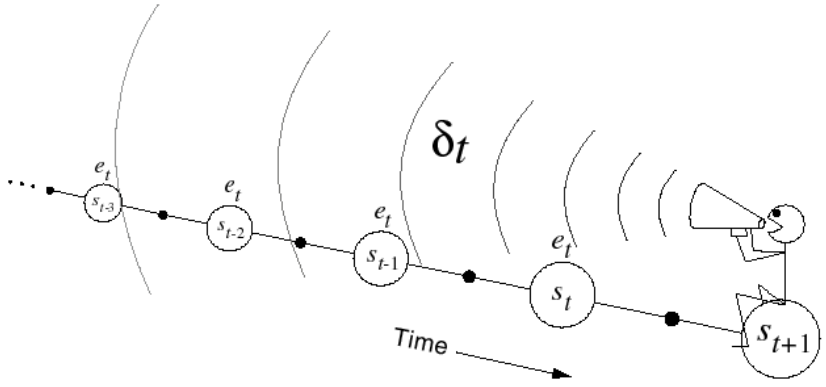
λ-return algorithm → TD(λ)

λ-return on the Random Walk



Same 19 state random walk as before

Backward View of TD(λ)



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

- Shout δ_t backwards over time
- The strength of your voice decreases with temporal distance by $\gamma\lambda$

Backward View of TD(λ)

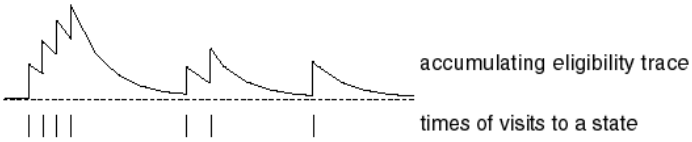
- The forward view was for theory
- The backward view is for mechanism

- New variable called *eligibility trace*. The eligibility trace for state at time is denoted

$$e_t(s) \in \Re^+$$

On each step, decay all traces by $\gamma\lambda$ and increment the trace for the current state by

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



γ discount rate

λ trace-decay parameter

Backward View of TD(λ)

The traces indicate the degree to which each state is eligible 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. For example, state-value prediction TD error:

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

one-step TD error

The global TD error signal triggers proportional updates to all recently visited states, as signaled by their nonzero traces:

$$\Delta V_t(s) = \alpha\delta_t e_t(s) \quad \forall s \in S$$

As always, these increments could be done on each step to form an on-line algorithm, or saved until the end of the episode to produce an off-line algorithm.

On-line Tabular TD(λ)

Initialize $V(s)$ arbitrarily
Repeat (for each episode):
  $e(s) = 0$, for all $s \in S$
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$
  Until s is terminal

Backward View – TD(0)

The backward view of TD(λ) is oriented backward in time. At each moment we look at the current TD error and assign it backward to each prior state according to the state's eligibility trace at that time.

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

$$\lambda = 0$$

$$e_t(s) = \begin{cases} 0 & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases}$$

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

$$\Delta V_t(s) = \alpha\delta_t e_t(s) \quad \forall s \in S$$

$$\Delta V_t(s) = \begin{cases} 0 & \text{if } s \neq s_t \\ \alpha \cdot \delta_t \cdot 1 & \text{if } s = s_t \end{cases}$$

The TD(λ) update reduces to the simple TD rule (TD(0)). Only the one state preceding the current one is changed by the TD error.

Backward View – TD(1) – MC

$\lambda = 1$

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

Similar to MC.

$\lambda = 1 = \gamma$

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

Similar to MC for an undiscounted episodic task.

If you set λ to 1, you get MC but in a better way

- Can apply TD(1) to continuing tasks
- Works incrementally and on-line (instead of waiting to the end of the episode)

Equivalence of Forward and Backward Views

- The forward (theoretical, λ-return algorithm) view of TD(λ) is equivalent to the backward (mechanistic) view for off-line updating
- The book shows:

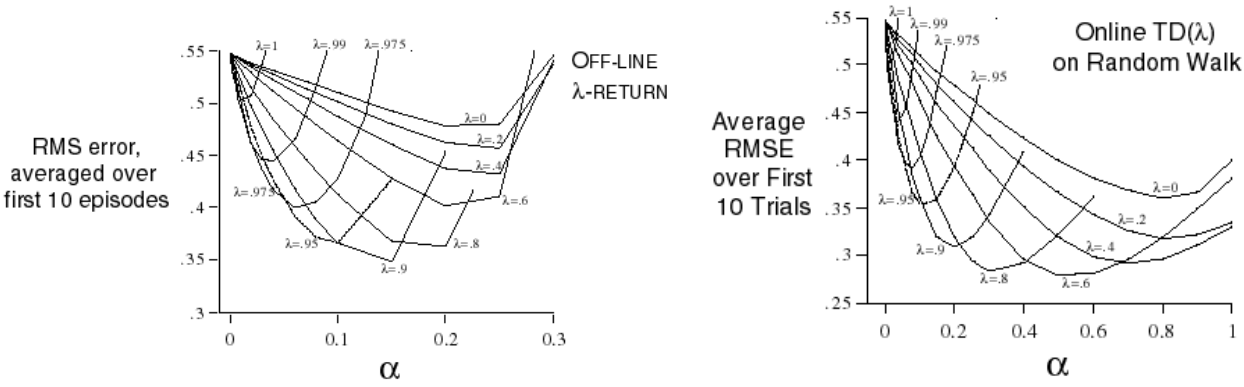
$$\underbrace{\sum_{t=0}^{T-1} \Delta V_t^{TD}(s)}_{\text{Backward updates}} = \underbrace{\sum_{t=0}^{T-1} \Delta V_t^{\lambda}(s_t) I_{ss_t}}_{\text{Forward updates}} \qquad I_{ss_t} = \begin{cases} s = s_t : 1 \\ else : 0 \end{cases}$$

algebra shown in book

$$\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) = \sum_{t=0}^{T-1} \alpha I_{ss_t} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \delta_k \qquad \sum_{t=0}^{T-1} \Delta V_t^{\lambda}(s_t) I_{ss_t} = \sum_{t=0}^{T-1} \alpha I_{ss_t} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \delta_k$$

On-line updating with small α is similar

On-line versus Off-line on Random Walk



- Same 19 state random walk
- On-line performs better over a broader range of parameters

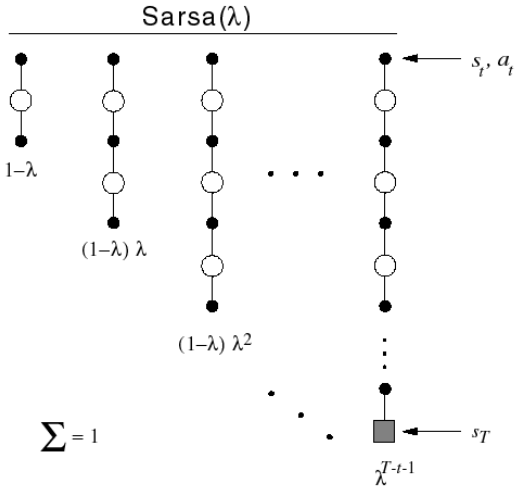
Sarsa(λ)

Eligibility-traces for state-action pairs:

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

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$$

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)$$



Triggers an update of all recently visited state-action values

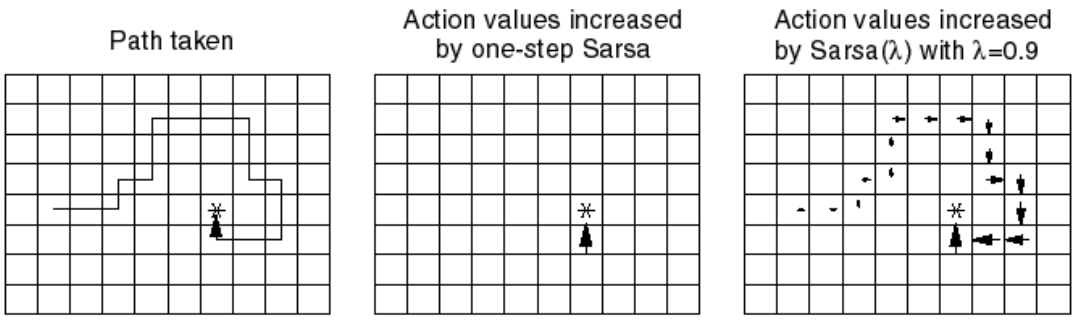
Sarsa(λ)

Algorithm:

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  $e(s,a) = 0$, for all s,a
  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. ε-greedy)
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s,a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'; a \leftarrow a'$
  Until s is terminal

Sarsa(λ)

Gridworld Example:



- With one trial, the agent has much more information about how to get to the goal
  - not necessarily the *best* way
- Can considerably accelerate learning

Q(λ)

A problem occurs for off-policy methods such as Q-learning when exploratory actions occur, since you backup over a non-greedy policy. This would violate GPI.

Three Approaches to Q(λ):

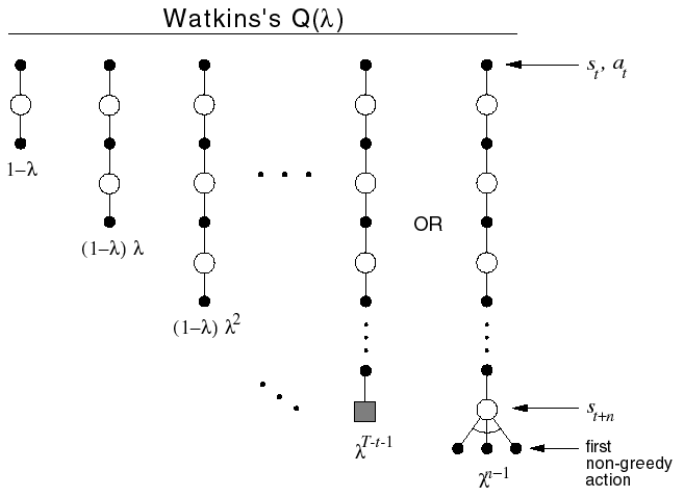
Watkins: Zero out eligibility trace after a non-greedy action. Do max when backing up at first non-greedy choice.

Peng: No distinction between exploratory and greedy actions.

Naïve: Similar to Watkins's method, except that the traces are not set to zero on exploratory actions.

Watkins's Q(λ)

Watkins: Zero out eligibility trace after a non-greedy action. Do max when backing up at first non-greedy choice.



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

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$$

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)$$

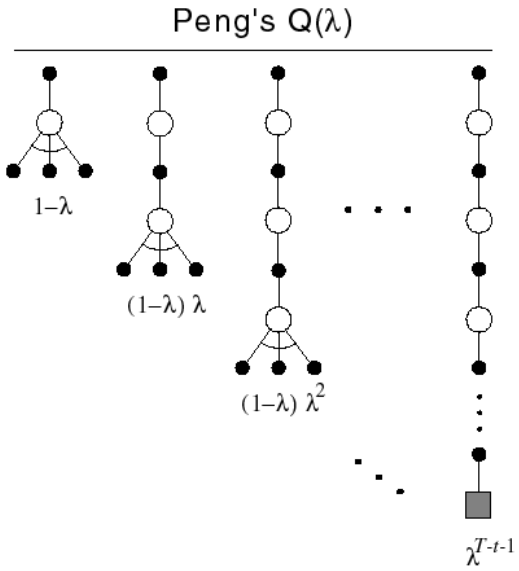
- Disadvantage to Watkins's method:
  - Early in learning, the eligibility trace will be "cut" frequently resulting in short traces

Watkins's Q(λ)

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  $e(s,a) = 0$, for all s,a
  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. ε-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(λ)

- No distinction between exploratory and greedy actions
- Backup max action except at end
- Never cut traces
- The earlier transitions of each are on-policy, whereas the last (fictitious) transition uses the greedy policy



- Disadvantage:
  - Complicated to implement
  - Theoretically, no guarantee to converge to the optimal value

Peng's Q(λ) Algorithm

Tabular version of Peng's Q(λ) algorithm

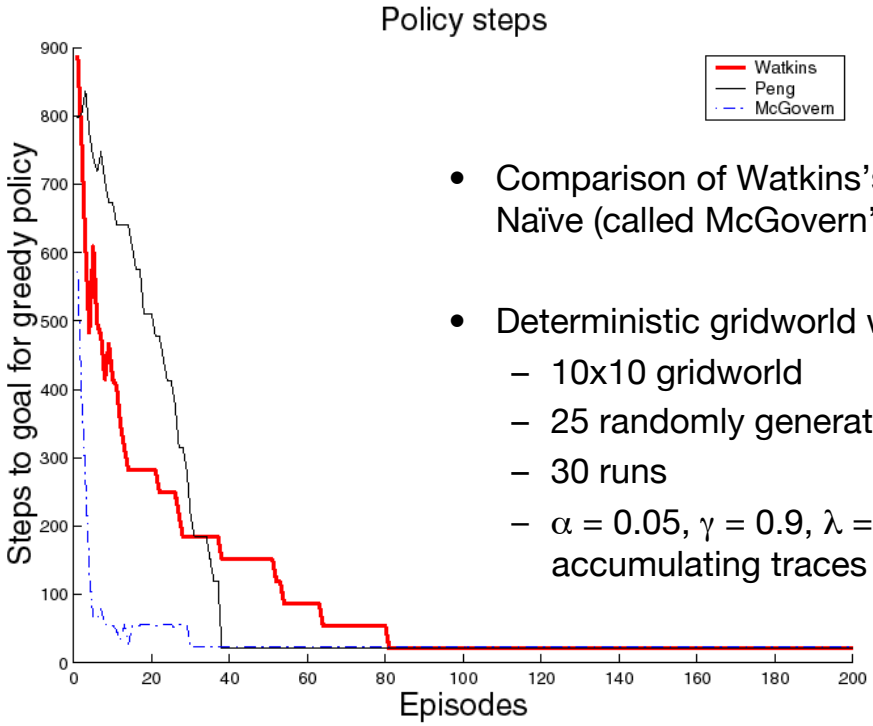
1. $Q(s,a) = 0$ and $e(s,a) = 0$ for all s,a
2. Do Forever:
   (a) $s_t \leftarrow$ the current state
   (b) Choose an action a_t according to current exploration policy
   (c) Carry out action a_t in the world. Let the short-term reward be r_t, and the new state s_{t+1}
   (d) $\delta'_t = r_t + \gamma V_t(s_{t+1}) - Q_t(s_t,a_t)$
   (e) $\delta_t = r_t + \gamma V_t(s_{t+1}) - V_t(s_t)$
   (f) For each state-action pair (s,a) do
   $e(s,a) = \gamma\lambda e(s,a)$
   $Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e(s,a)$
   (g) $Q_{t+1}(s_t,a_t) = Q_{t+1}(s_t,a_t) + \alpha\delta'_t$
   $e(s_t,a_t) \leftarrow e(s_t,a_t) + 1$

For a complete description of the needed implementation, see Peng & Williams (1994, 1996).

Naive Q(λ)

- Idea: is it really a problem to backup exploratory actions?
  - Never zero traces
  - Always backup max at current action (unlike Peng or Watkins's)
- Works well in preliminary empirical studies

Comparison Results



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

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

Convergence of the Q(λ)'s

- 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^*
- Naive - Q^*?

Eligibility Traces for Actor-Critic Methods

- *Critic:* On-policy learning of V^{π}. Use TD(λ) as described before.
- *Actor:* Needs eligibility traces for each state-action pair.

We change the update equation of the actor:

$$p_{t+1}(s,a) = \begin{cases} p_t(s,a) + \alpha\delta_t & \text{if } a = a_t \text{ and } s = s_t \\ p_t(s,a) & \text{otherwise} \end{cases} \quad \text{to} \quad p_{t+1}(s,a) = p_t(s,a) + \alpha\delta_t e_t(s,a)$$

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

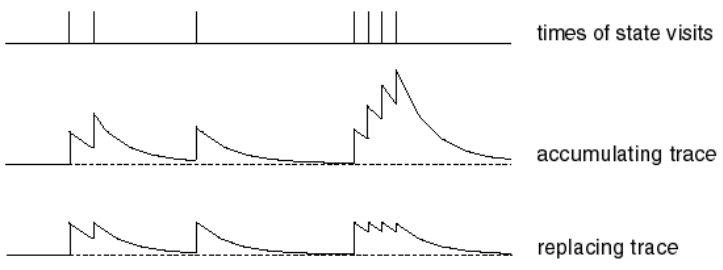
Replacing Traces

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

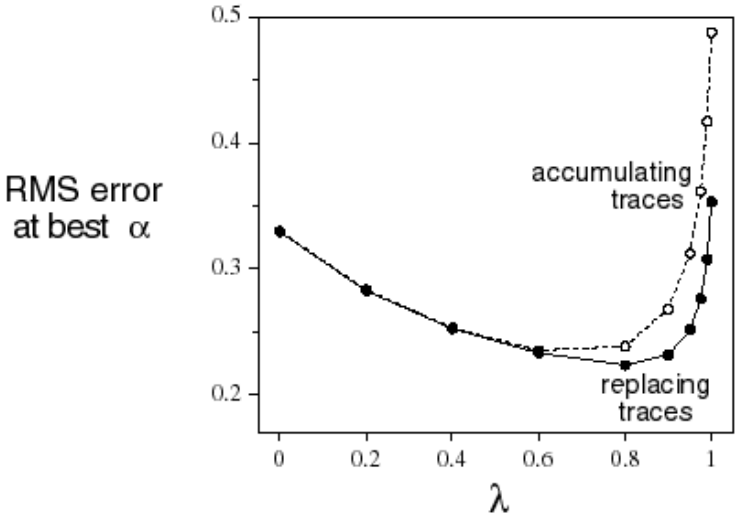
$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases}$$



Replacing Traces Example

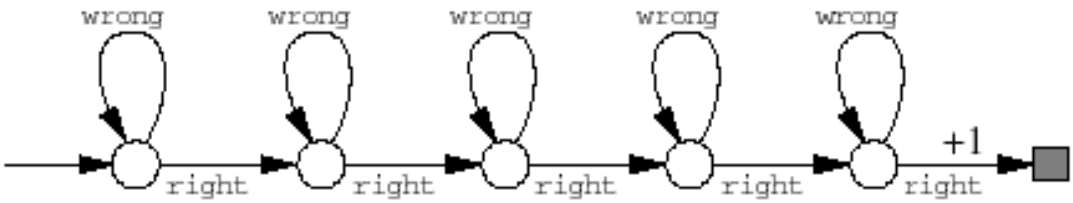
Same 19 state random walk task as before

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



Why Replacing Traces?

- Replacing traces can significantly speed learning
- They can make the system perform well for a broader set of parameters
- Accumulating traces can do poorly on certain types of tasks



Why is this task particularly onerous for accumulating traces?

More Replacing Traces

There is an interesting relationship between replace-trace methods and Monte Carlo methods in the undiscounted case:

Just as conventional TD(1) is related to the every-visit MC algorithm, off-line replace-trace TD(1) is identical to first-visit MC (Singh and Sutton, 1996).

Extension to action-values:

When you revisit a state, what should you do with the traces for the other actions?

Singh & Sutton (1996) proposed to set them to zero:

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

Variable λ

Can generalize to variable λ

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

Here λ is a function of time

Could define

$$\lambda_t = \lambda(s_t) \text{ or } \lambda_t = \lambda^{t/\tau}$$

Conclusions

- Provides efficient, incremental way to combine MC and TD
  - Includes advantages of MC (can deal with lack of Markov property)
  - Includes advantages of TD (using TD error, bootstrapping)
- Can significantly speed learning
- Does have a cost in computation

References

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