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

Relation of Backwards View to MC & TD(0)

- Using update rule:

$$\Delta V_t(s) = \alpha \delta_t e_t(s)$$

- As before, if you set λ to 0, you get to TD(0)
- 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)

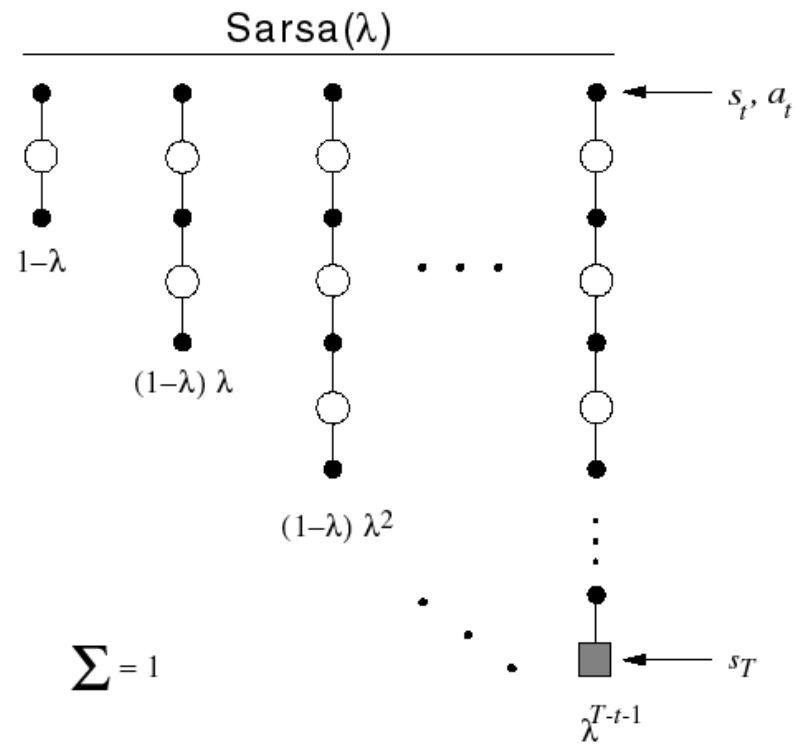
Control: Sarsa(λ)

- Save eligibility for state-action pairs instead of just states

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

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

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



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:

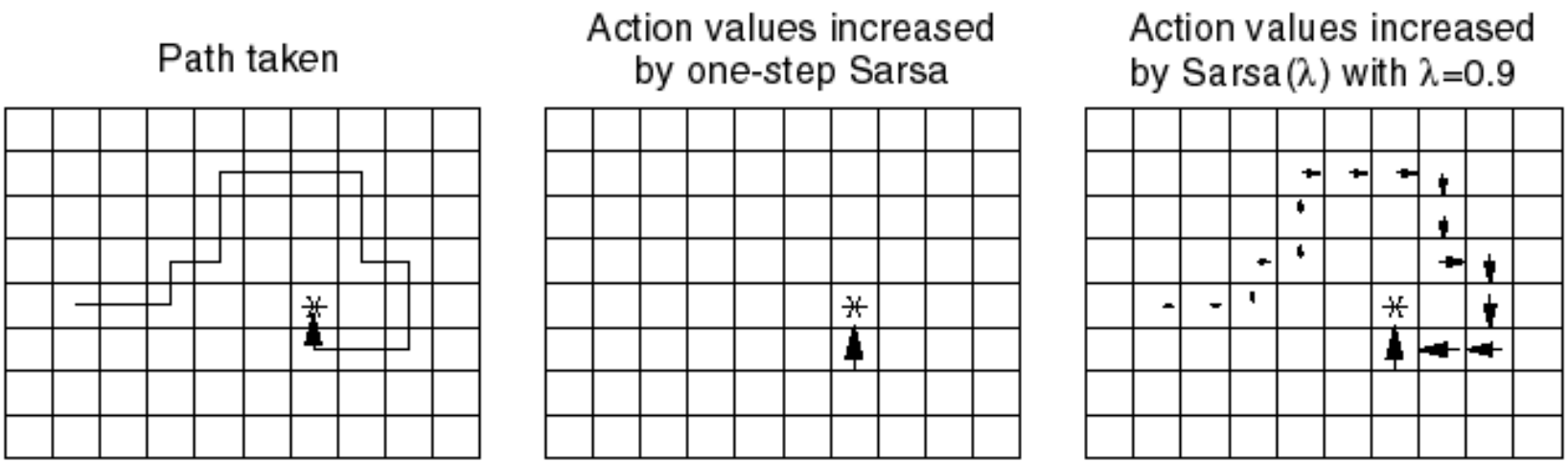
$Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$

$e(s,a) \leftarrow \gamma\lambda e(s,a)$

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

Until s is terminal

Sarsa(λ) Gridworld Example



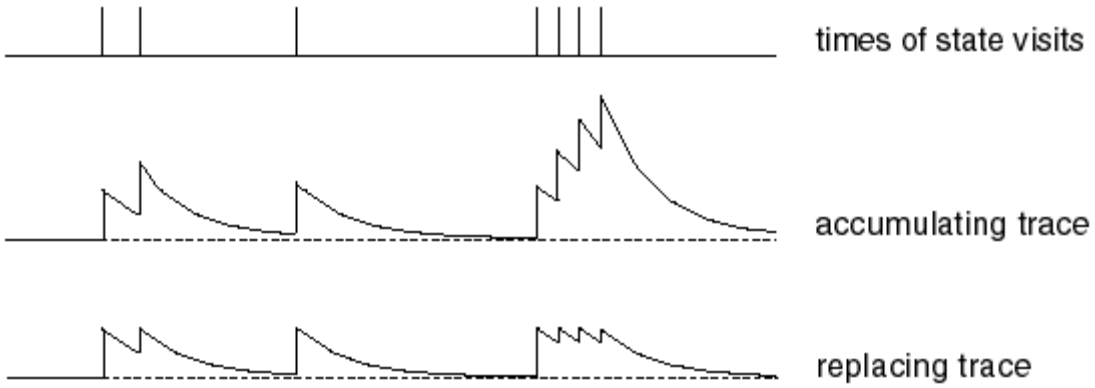
- With one trial, the agent has much more information about how to get to the goal
  - not necessarily the *best* way
- Can considerably accelerate learning

Replacing Traces

- Using accumulating traces, frequently visited states can have eligibilities greater than 1
  - This can be a problem for convergence

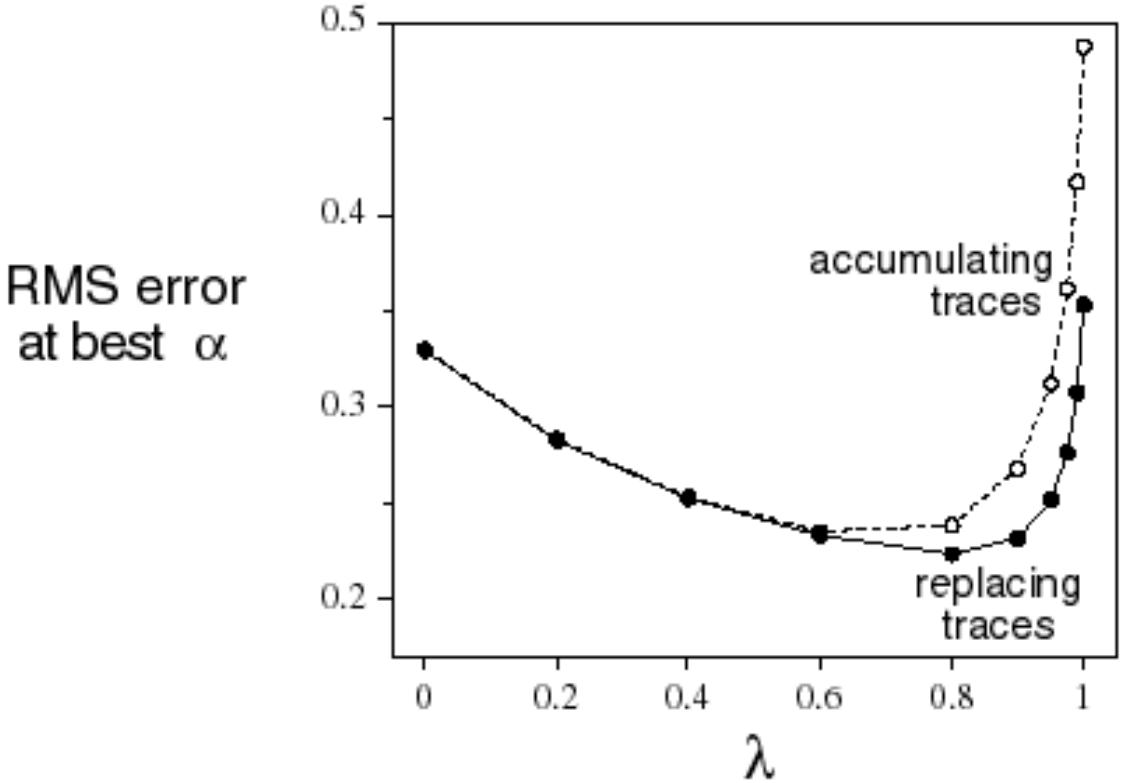
- *Replacing traces:* Instead of adding 1 when you visit a state, set that trace to 1

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



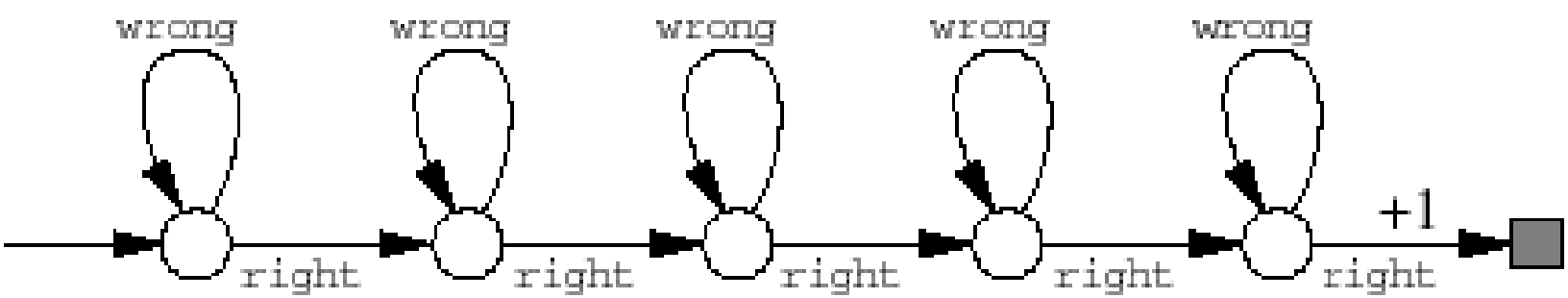
Replacing Traces Example

- 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

- Off-line replacing trace TD(1) is identical to first-visit MC

- Extension to action-values:
  - When you revisit a state, what should you do with the traces for the other actions?
  - Singh and Sutton say to set them to zero:

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

Implementation Issues with Traces

- Could require much more computation
  - But most eligibility traces are VERY close to zero
- If you implement it in Matlab, backup is only one line of code and is very fast (Matlab is optimized for matrices)

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

Chapter 8: Function approximation

Any FA Method?

- In principle, yes:
  - artificial neural networks
  - decision trees
  - multivariate regression methods
  - etc.
- But RL has some special requirements:
  - usually want to learn while interacting
  - ability to handle nonstationarity
  - other?