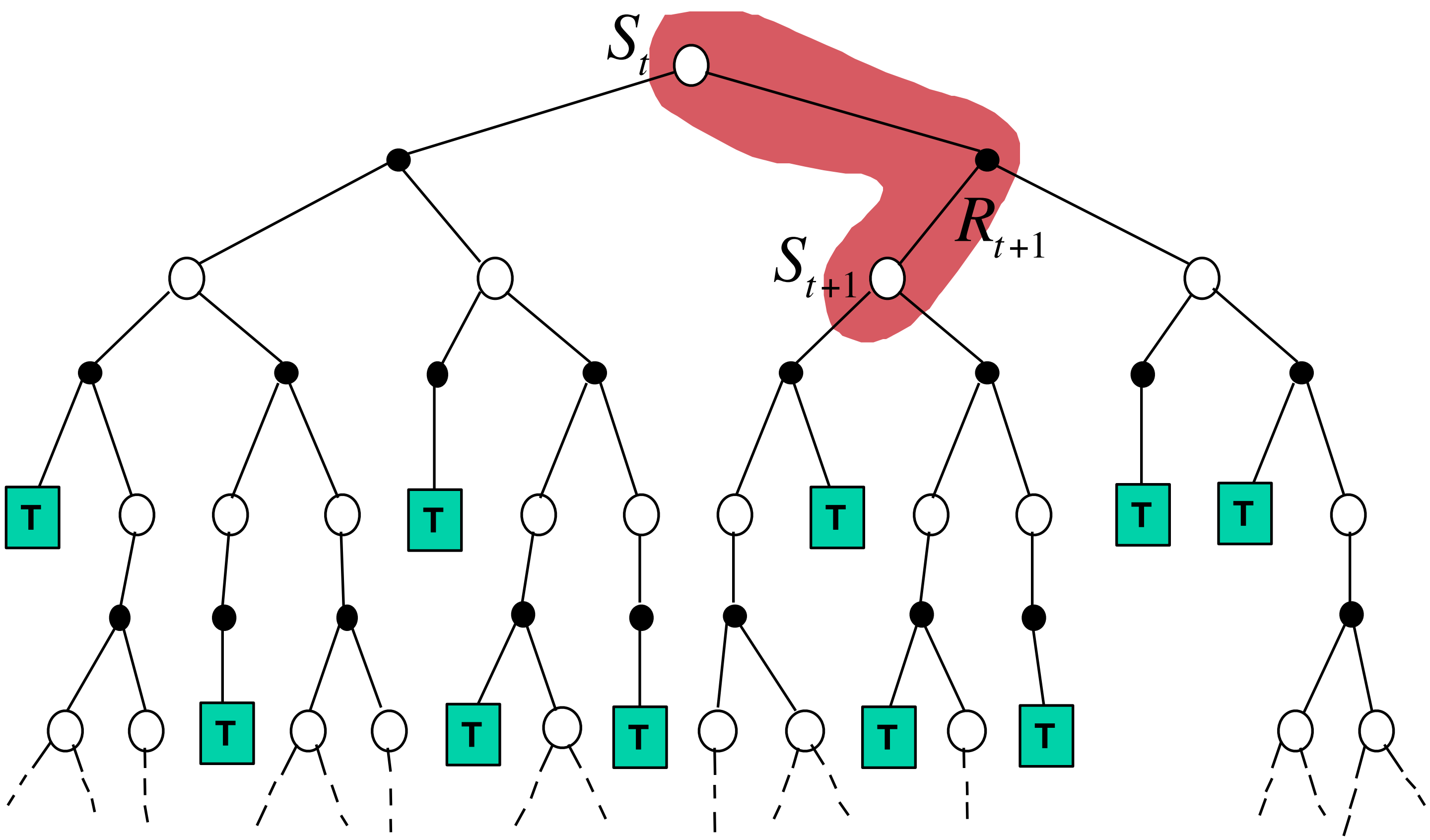
Lecture 8: More on Temporal-Difference Learning. Eligibility traces.

Recall: Temporal-Difference Learning: Between MC and DP!

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$



Temporal-Difference (TD) Prediction

Policy Evaluation (the prediction problem):

for a given policy π, compute the state-value function v_π

Simple every-visit Monte Carlo method:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

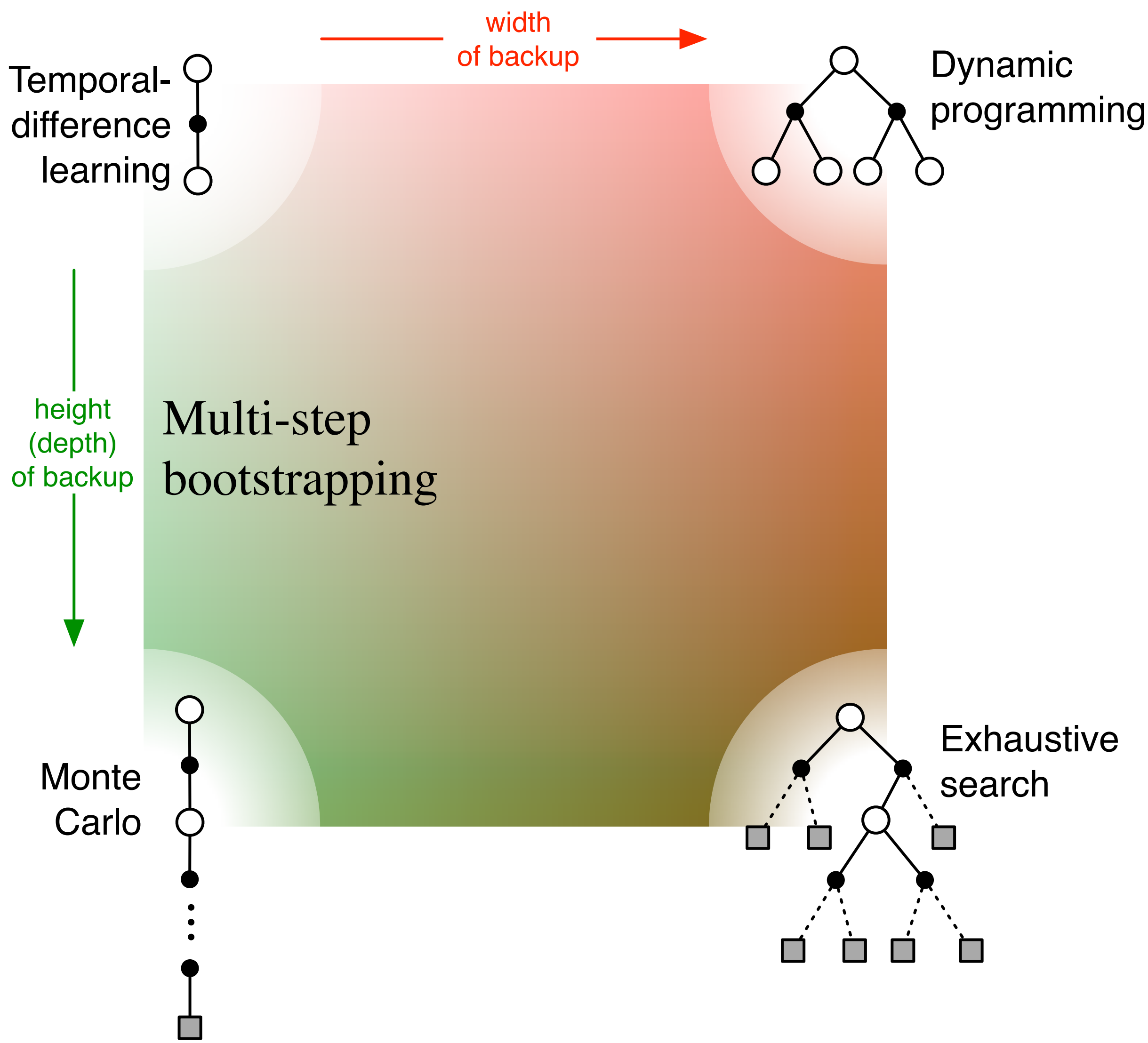
target: the actual return after time *t*

The simplest temporal-difference method TD(0):

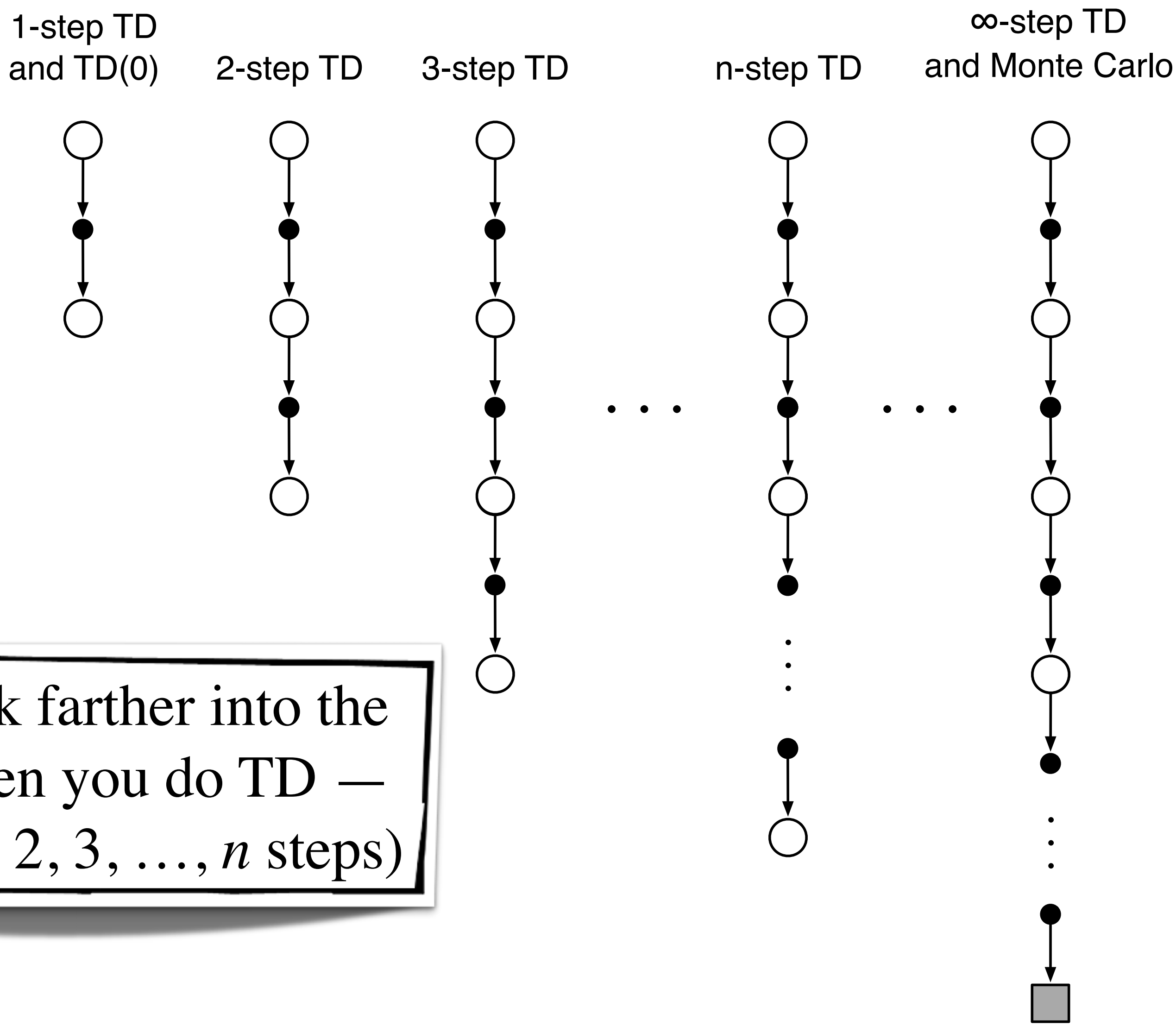
$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

target: an estimate of the return

Unified View



n-step TD Prediction



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

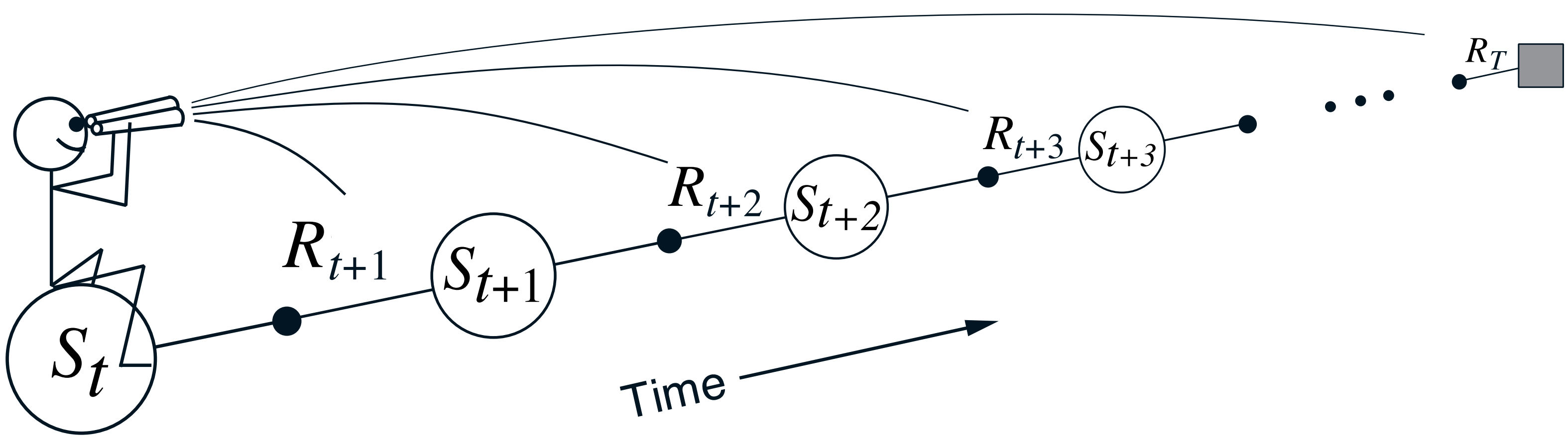
Mathematics of *n*-step TD Returns/Targets

- Monte Carlo: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$
- TD: $G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$
  - Use V_t to estimate remaining return
- *n*-step TD:
  - 2 step return: $G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_t(S_{t+2})$
  - *n*-step return: $G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_t(S_{t+n})$

    with $G_t^{(n)} \doteq G_t$ if $t + n \geq T$

Forward View

- Look forward from each state to determine update from future states and rewards:



n-step TD

- Recall the n-step return:

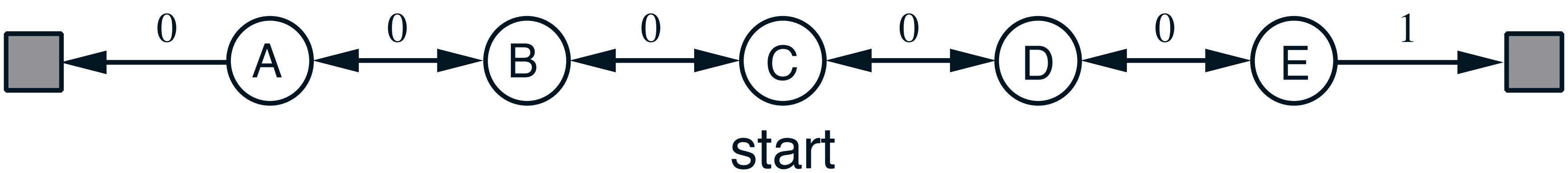
$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \quad n \geq 1, 0 \leq t < T-n$$

- Of course, this is <u>not available</u> until time $t+n$
- The natural algorithm is thus to wait until then:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_t^{(n)} - V_{t+n-1}(S_t) \right], \qquad 0 \leq t < T,$$

- $w_{t+1} \leftarrow w_t + \alpha(G_t^{(n)} - V_w(S_t)) \nabla_w V_w(S_t)$, with FA
- This is called n-step TD

Random Walk Examples



- Suppose the trajectory is $C \rightarrow D \rightarrow E \rightarrow T$
- How does 2-step TD work here?
- How about 3-step TD?

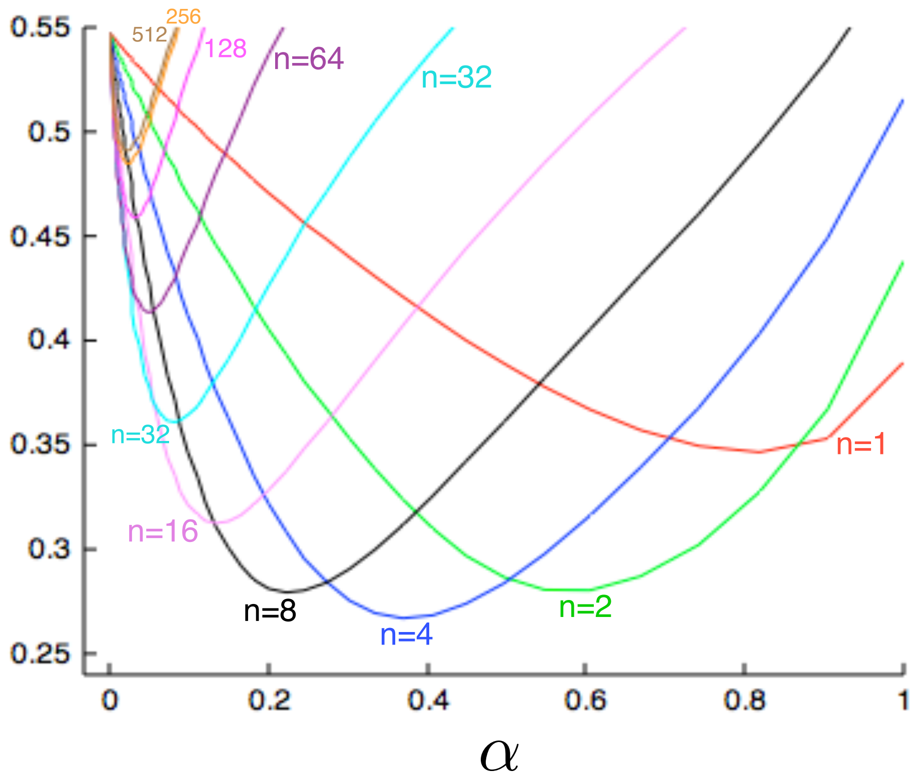
n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

Repeat (for each episode):
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    For $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | Take an action according to $\pi(\cdot|S_t)$
    | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | If S_{t+1} is terminal, then $T \leftarrow t+1$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
    | If $\tau \geq 0$:
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ ($G_\tau^{(n)}$)
    | $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
    Until $\tau = T - 1$

A Larger Example – 19-state Random Walk

Average RMS error over 19 states and first 10 episodes



n-step TD results

- An intermediate α is best
- An intermediate n is best
- Do you think there is an optimal n? for every task?

Recall: RL with function approximation

General SGD: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} Error_t^2$

For VFA: $\leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta})\right]^2$

Chain rule: $\leftarrow \boldsymbol{\theta} - 2\alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta})\right] \nabla_{\boldsymbol{\theta}} \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta})\right]$

Semi-gradient: $\leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta})\right] \nabla_{\boldsymbol{\theta}} \hat{v}(S_t, \boldsymbol{\theta})$

Linear case: $\leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{v}(S_t, \boldsymbol{\theta})\right] \boldsymbol{\phi}(S_t)$

Action-value form: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left[Target_t - \hat{q}(S_t, A_t, \boldsymbol{\theta})\right] \boldsymbol{\phi}(S_t, A_t)$

Different algorithms: Different Targets!

- Monte Carlo: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$
- TD: $G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$
  - Use V_t to estimate remaining return
- n-step TD:
  - 2 step return: $G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_t(S_{t+2})$
  - n-step return: $G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2$ [illegible] $+ \gamma^{n-1} R_{t+n} + \gamma^n V_t(S_{t+n})$

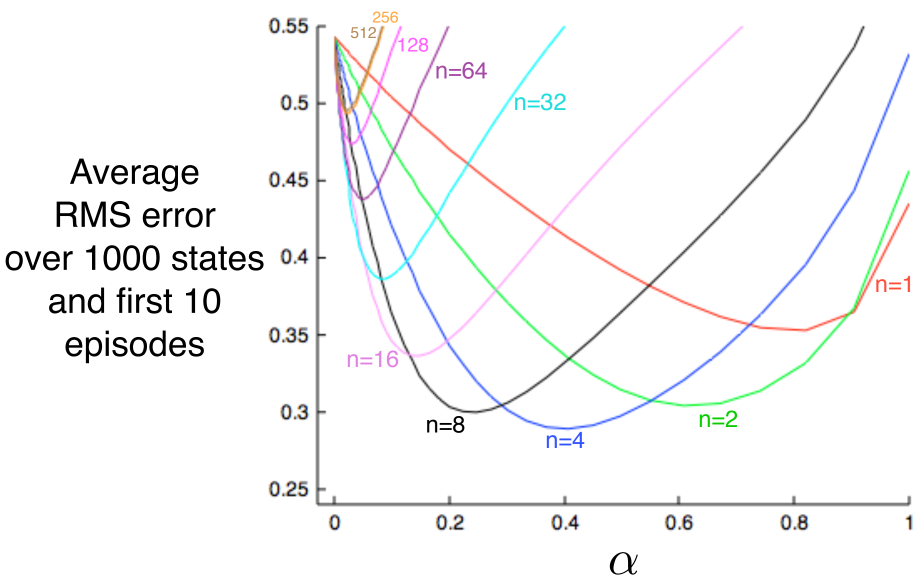
    $G_t^{(n)} \doteq G_t \text{ if } t + n \geq T$

n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

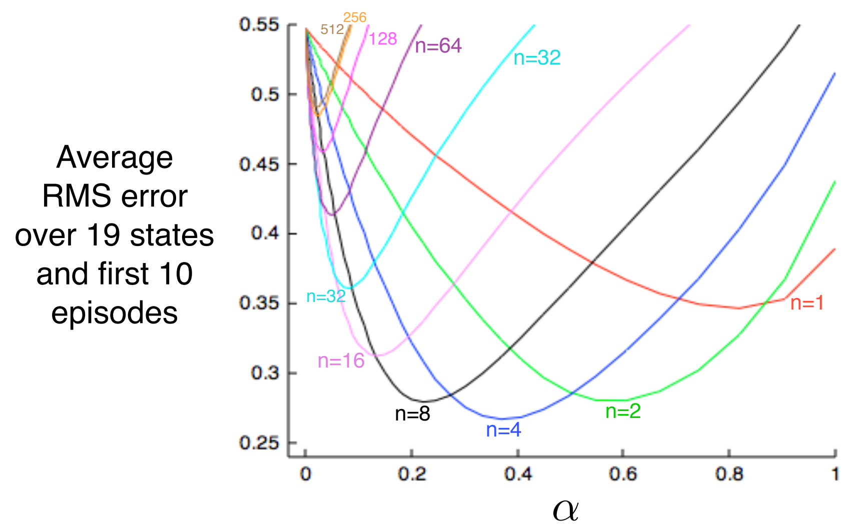
Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, a positive integer n
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t and R_t) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | Take an action according to $\pi(\cdot|S_t)$
    | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | If S_{t+1} is terminal, then $T \leftarrow t + 1$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
    | If $\tau \geq 0$:
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n \hat{v}(S_{\tau+n},\mathbf{w})$ ($G_{\tau:\tau+n}$)
    | $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{v}(S_\tau,\mathbf{w})\right] \nabla \hat{v}(S_\tau,\mathbf{w})$
    Until $\tau = T - 1$

Bootstrapping also speeds learning with FA



1000 states aggregated
into 20 groups of 50



19 states tabular

Conclusions Regarding *n*-step Methods (so far)

- Generalize Temporal-Difference and Monte Carlo learning methods, sliding from one to the other as n increases
  - $n = 1$ is TD(0) $n = \infty$ is MC
  - an intermediate n is often much better than either extreme
  - applicable to both continuing and episodic problems
- There is some cost in computation
  - need to remember the last n states
  - learning is delayed by n steps
  - per-step computation is small and uniform, like TD

Eligibility Traces

- Another way of interpolating between MC and TD methods
- A way of implementing *compound λ-return* targets
- A basic mechanistic idea — a short-term, fading memory
- A new style of algorithm development/analysis
  - the forward-view ⇔ backward-view transformation
  - Forward view:
    conceptually simple — good for theory, intuition
  - Backward view:
    computationally congenial implementation of the f. view

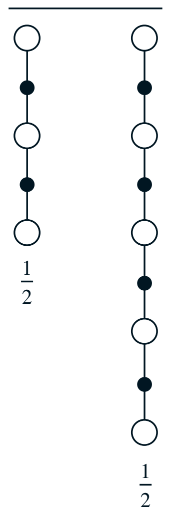
Any set of update targets can be *averaged* to produce new *compound* update targets

- For example, half a 2-step plus half a 4-step

$$U_t = \frac{1}{2}G_t^{(2)} + \frac{1}{2}G_t^{(4)}$$

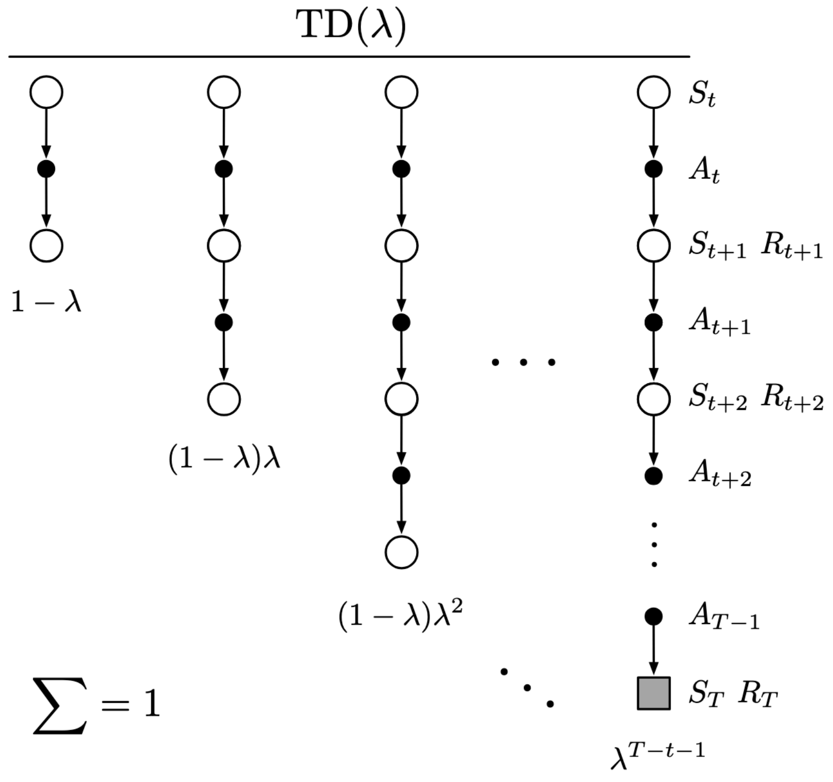
- Called a compound backup
  - Draw each component
  - Label with the weights for that component

A *compound* backup



The λ-return is a compound update target

- The λ-return a target that averages all n-step targets
- Each weighted by λ^{n-1}



$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{T-t-1}G_t,$$

Relation to TD(0) and MC

- The λ-return can be rewritten as:

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

- If λ = 1, you get the MC target:

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

- If λ = 0, you get the TD(0) target:

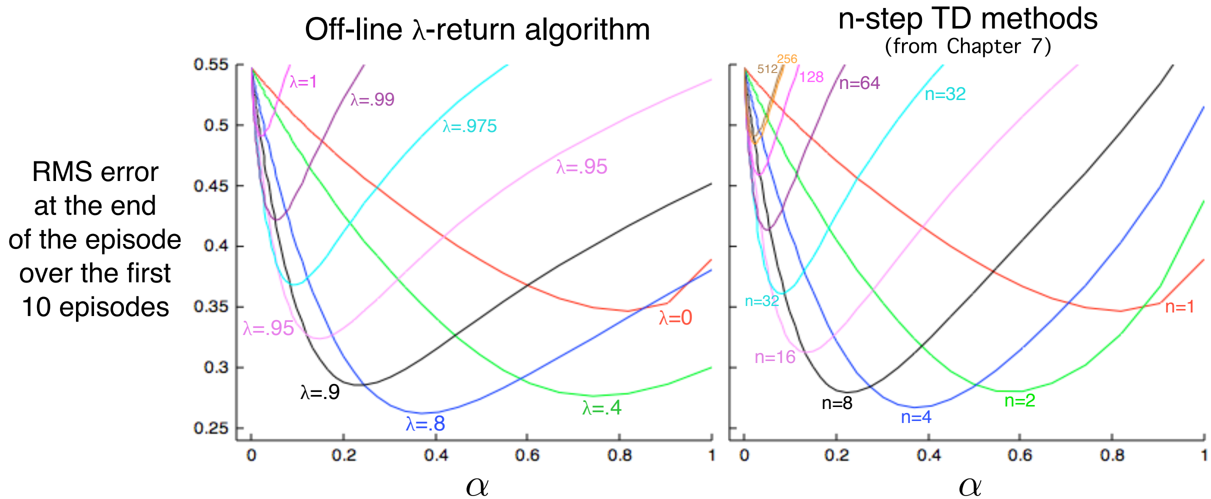
$$G_t^\lambda = (1-0)\sum_{n=1}^{T-t-1}0^{n-1}G_t^{(n)} + 0^{T-t-1}G_t = G_t^{(1)}$$

The off-line λ-return "algorithm"

- Wait until the end of the episode (offline)
- Then go back over the time steps, updating

$$w_{t+1} \leftarrow w_t + \alpha(G_t^n - V_w(S_t))\, \nabla_w V_w(S_t)$$

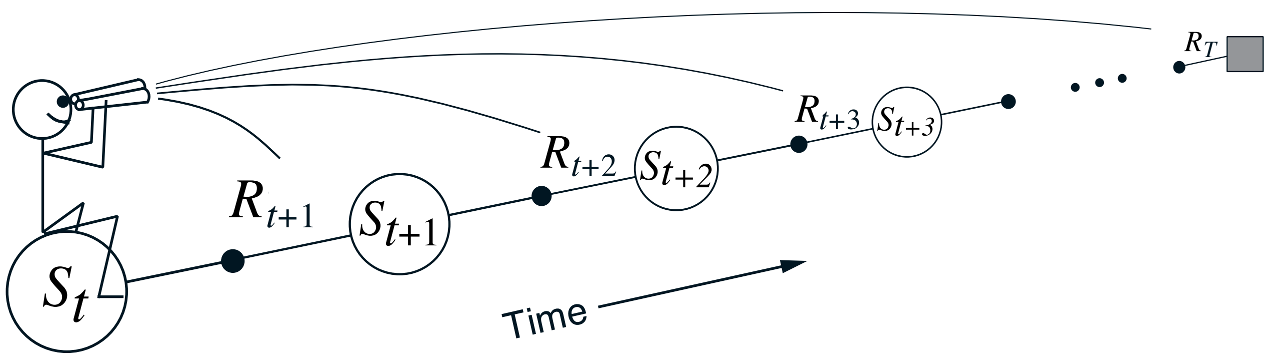
The λ-return alg performs similarly to *n*-step on the 19-state random walk (Tabular)



Intermediate λ is best (just like intermediate *n* is best)
λ-return slightly better than *n*-step

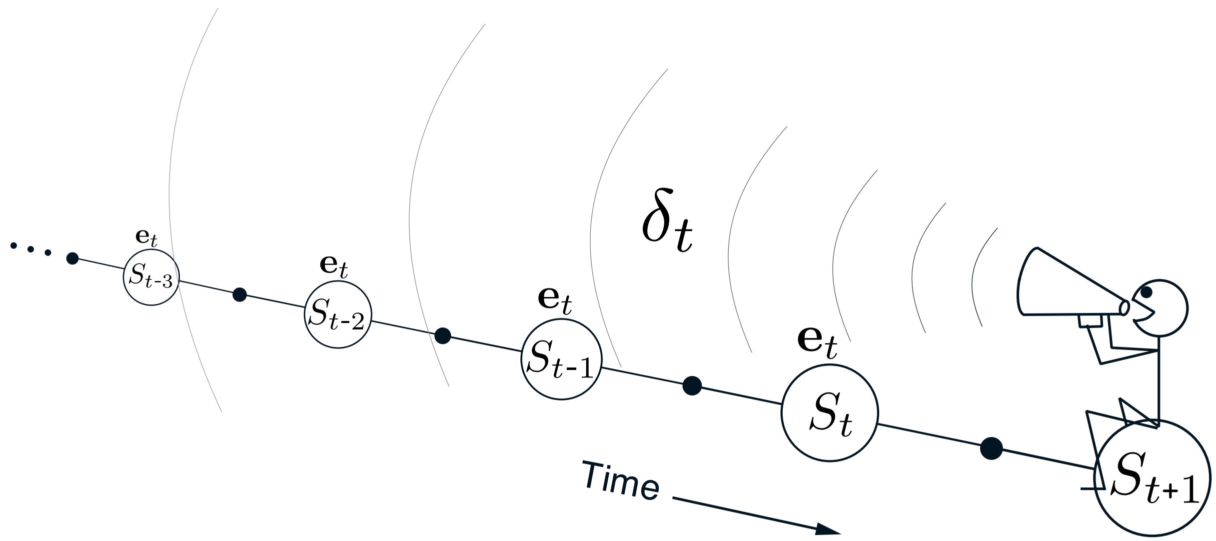
The forward view

- Look forward from each state to determine update from future states and rewards:



The backward view

- Shout the TD error backwards
- The traces fade with temporal distance by γλ



Eligibility traces (mechanism)

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

same shape as $\boldsymbol{\theta}$

- New memory vector called *eligibility trace* $\mathbf{e}_t \in \mathbb{R}^n \geq \mathbf{0}$
  - On each step, decay each component by γλ and increment the trace for the current state by 1
  - *Accumulating trace*

$$\mathbf{e}_0 \doteq \mathbf{0},$$
$$\mathbf{e}_t \doteq \nabla \hat{v}(S_t,\boldsymbol{\theta}_t) + \gamma\lambda\mathbf{e}_{t-1}$$

  - *Replacing trace:* trace becomes 1 when state is visited

The Semi-gradient TD(λ) algorithm

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \delta_t \mathbf{e}_t$$

$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1},\boldsymbol{\theta}_t) - \hat{v}(S_t,\boldsymbol{\theta}_t)$$

$$\mathbf{e}_0 \doteq \mathbf{0},$$
$$\mathbf{e}_t \doteq \nabla \hat{v}(S_t,\boldsymbol{\theta}_t) + \gamma\lambda \mathbf{e}_{t-1}$$

Online TD(λ)

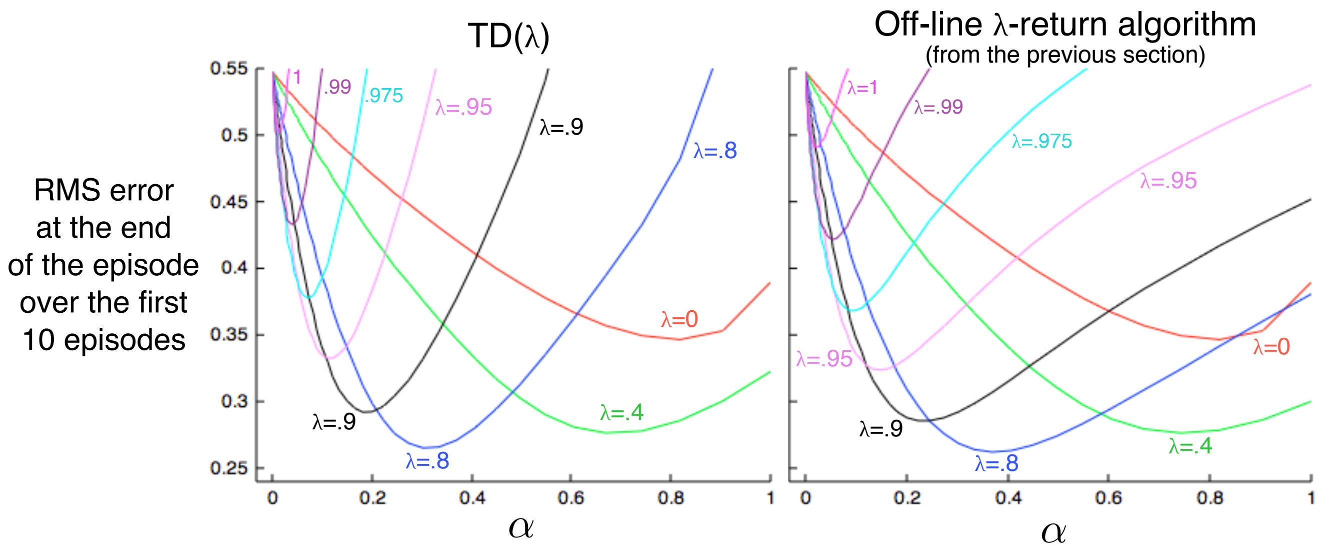
Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

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

TD(λ) performs similarly to offline λ-return but slightly worse, particularly at high α

Tabular 19-state random walk task



TD(λ) allows online updating!!

Summary: TD-family for policy evaluation

- The TD family of methods is between MC and DP
- Interpolating in terms of credit assignment length!
- With bootstrapping (TD), we don’t get true gradient descent methods with function approximation
  - this complicates the analysis
  - but learning is can be *much faster*