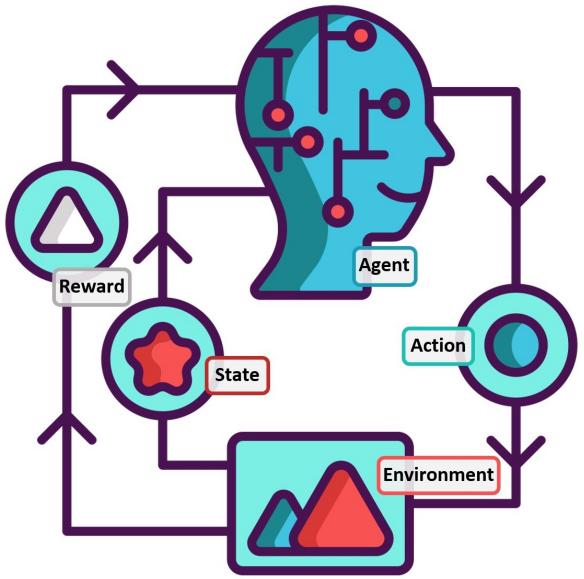
AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning
2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Lecture #12
n-step Bootstrapping & TD(λ)

Gian Antonio Susto



MC vs. TD(0)

MC: no bias

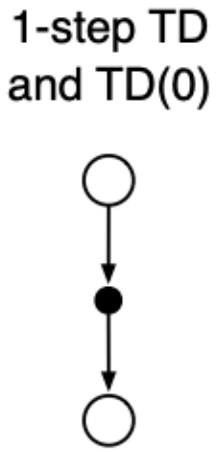
TD(0): low variance

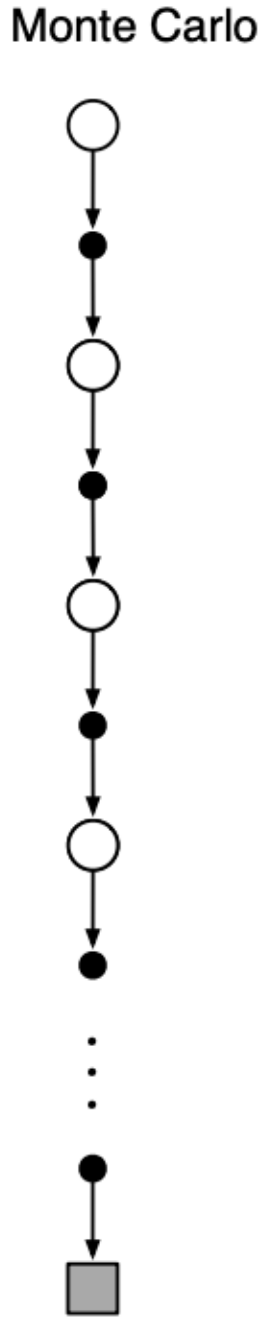
MC uses real return

TD(0) just rewards and bootstrap

MC updates just at the end of the episode

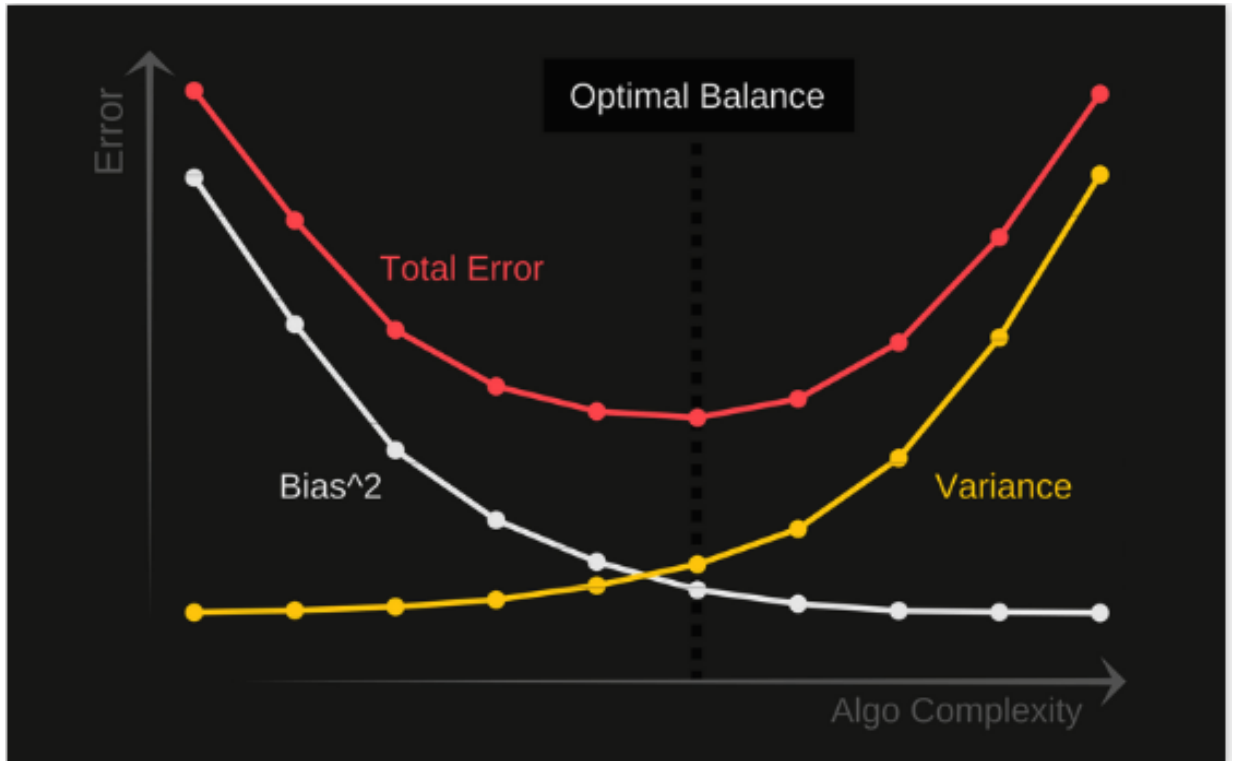
TD(0) updates along the way

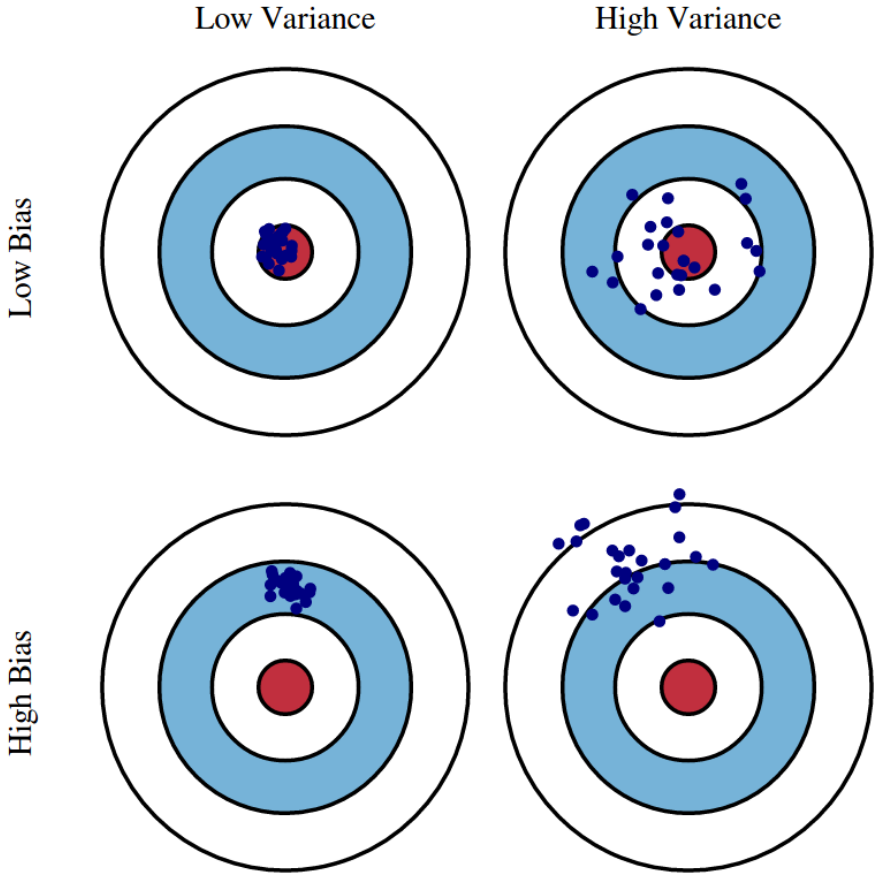




Bias/Variance Trade-off

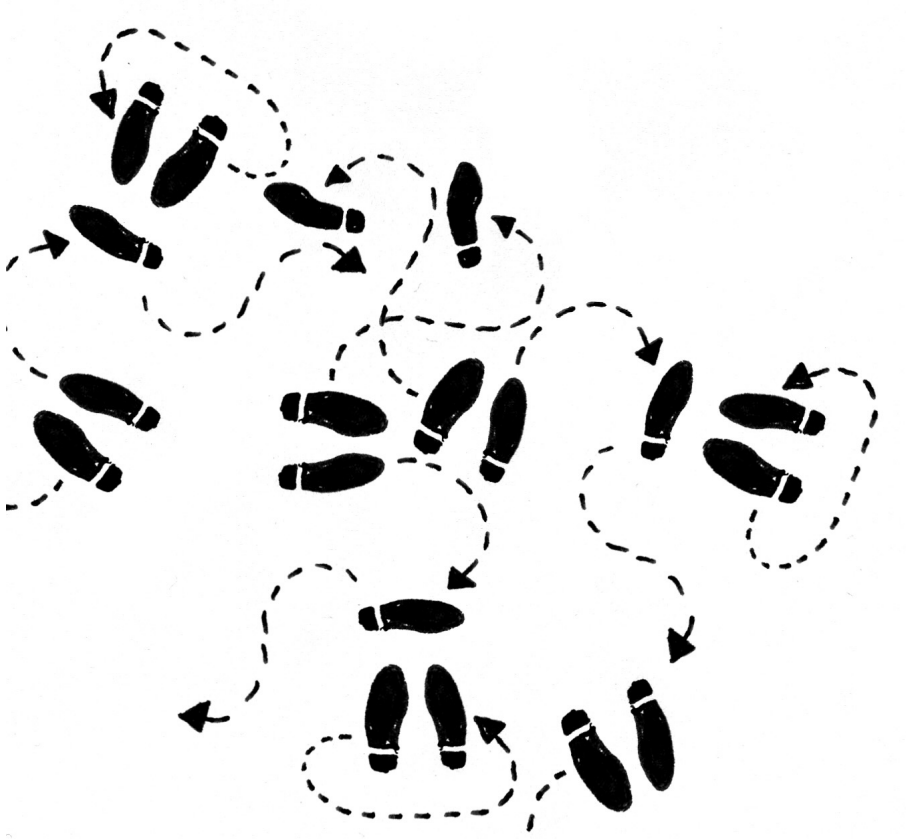
- A trade-off present in many Machine Learning settings



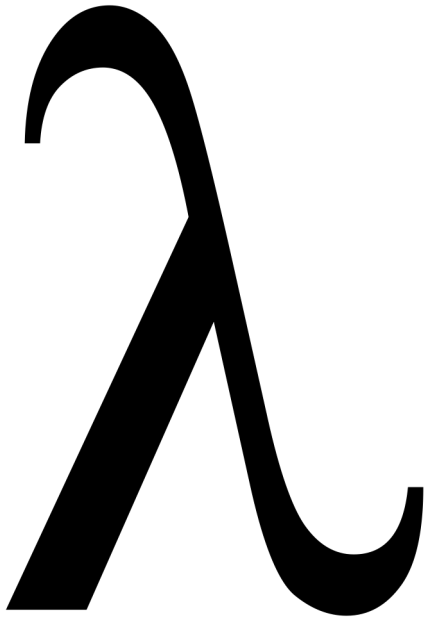


Approaches to balance Bias/Variance

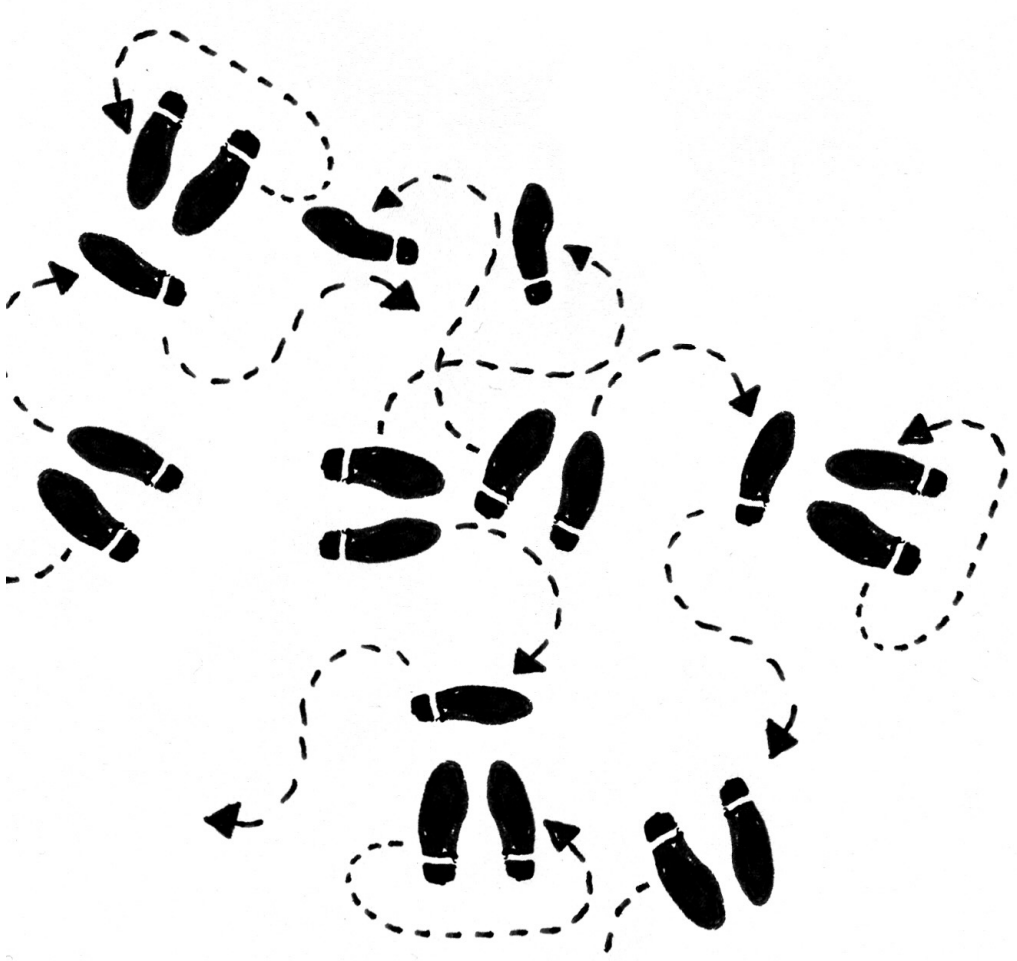
n-step bootstrapping - chapter 7



TD(λ) - chapter 12

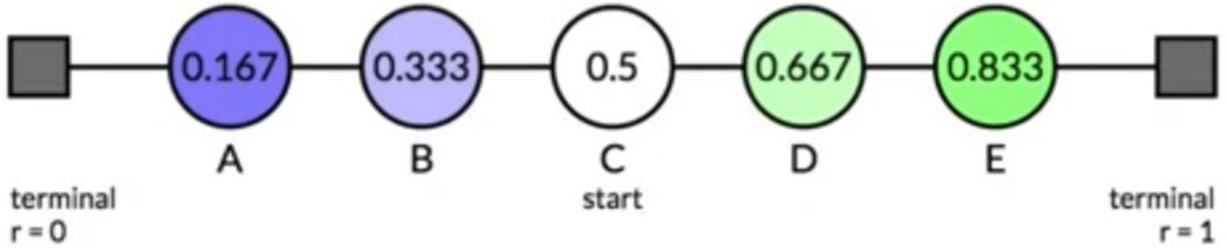


n-step Bootstrapping (Chapter 7)



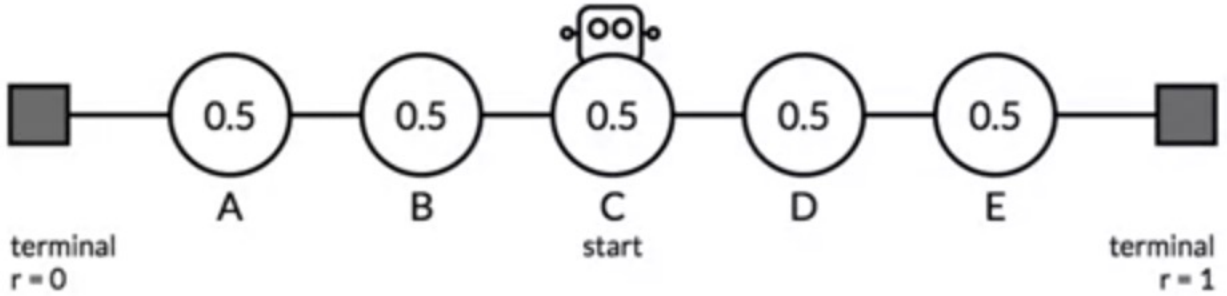
Episode #1

Target / Exact Values



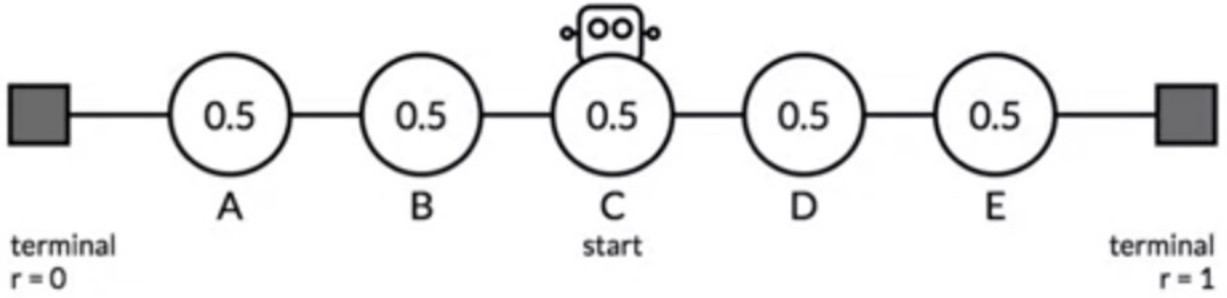
Updates using TD Learning

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



In the following:
$\alpha = 0.5$

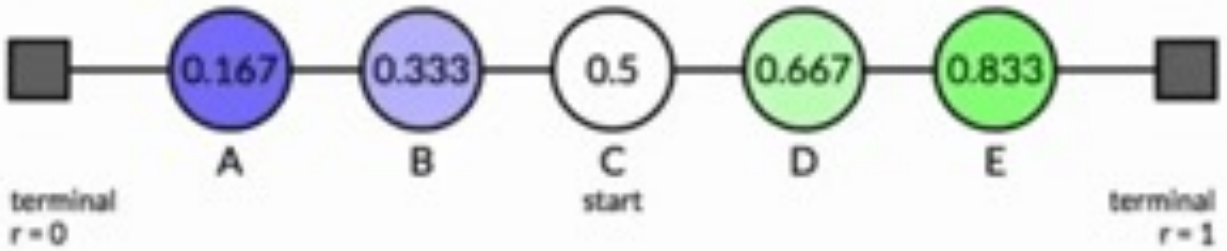
Updates using Monte Carlo



A. White, M. White 'Sample-based Learning Methods'

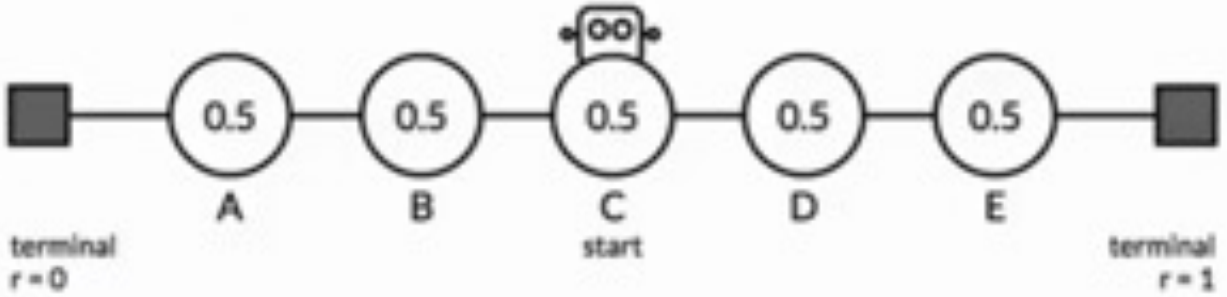
Episode #1

Target / Exact Values

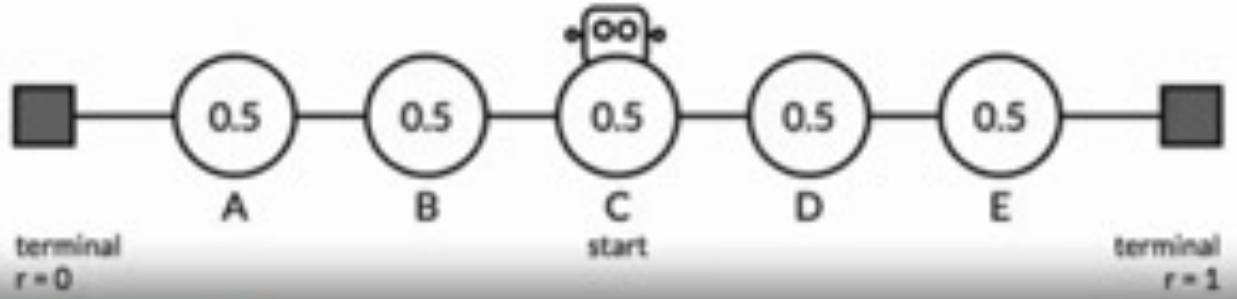


Updates using TD Learning

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



Updates using Monte Carlo



A. White, M. White 'Sample-based Learning Methods'

Prediction: MC vs. TD(0)

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

MC: $$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$

TD(0): $$V(S_t) \leftarrow V(S_t) + \alpha\,(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

The TD target can be seen as the 'best' estimation of G_t at time t+1:

$$G_t = R_{t+1} + \gamma G(S_{t+1}) \sim R_{t+1} + \gamma V(S_{t+1})$$

Prediction: why not considering a different estimation of G, for example after 2 steps?

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

$n = 1$ $\quad G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

$n = 2$ $\quad G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$

Prediction: why not considering a different estimation of G, for example after 2 steps?

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

This is the best estimation of *G* at time *t+2*!

$n = 1$ $\quad G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

$n = 2$ $\quad G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$

Prediction: why not considering a different estimation of G, for example after 2 steps?

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

This is the best estimation of *G* at time *t+2*!

$n = 1$ $\quad G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

$n = 2$ $\quad G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$

I can use this quantity as a different *TD target*

Prediction: why not considering a different estimation of G, for example after 2 steps?

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

This is the best estimation of *G* at time *t+2*!

$n = 1$ $\quad G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

$n = 2$ $\quad G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$

Pay attention to the notation!

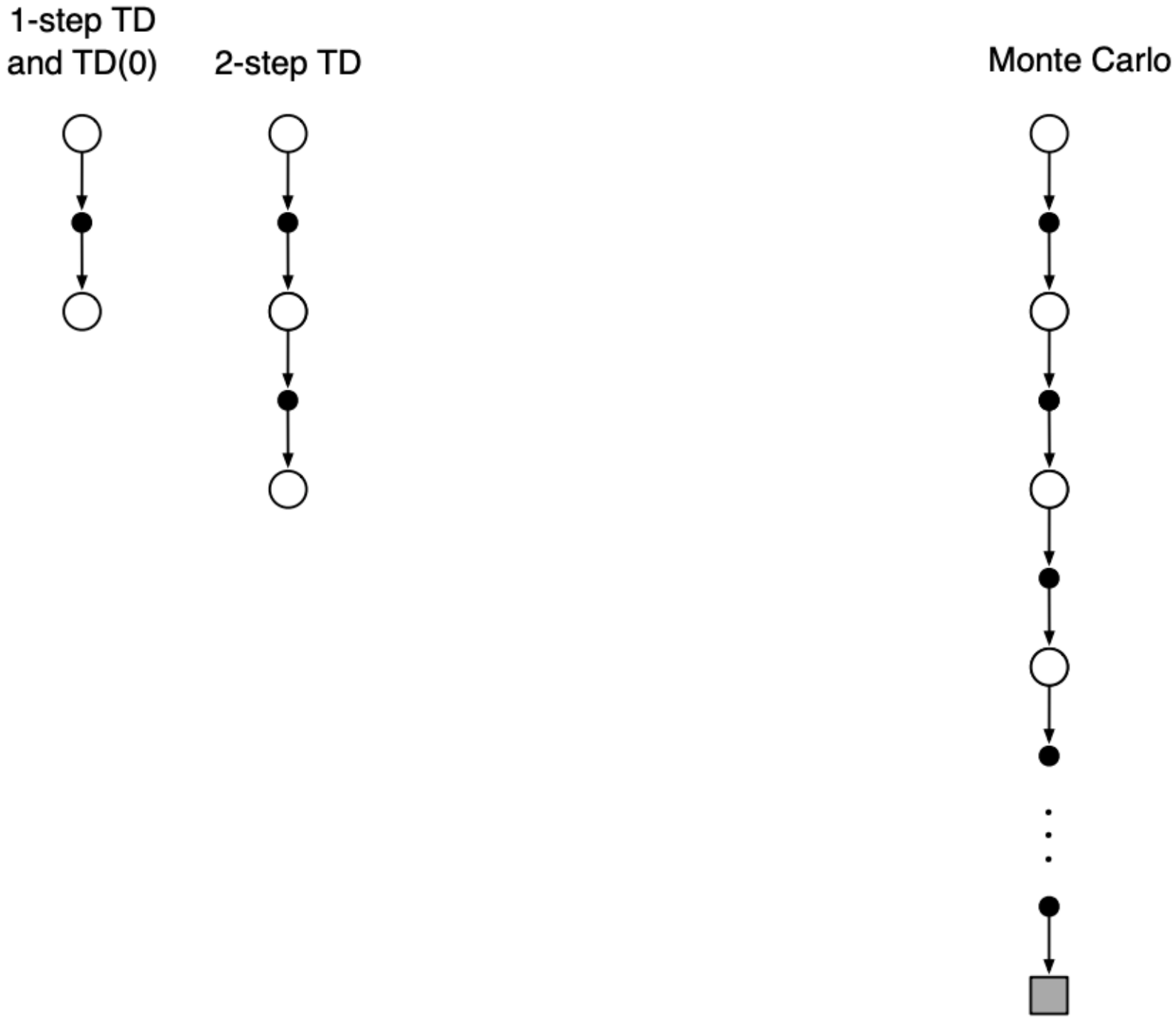
I can use this quantity as a different *TD target*

MC vs. TD(0)

MC: low bias

TD(0): low variance

Why don’t using a compromise between MC and TD(0)?

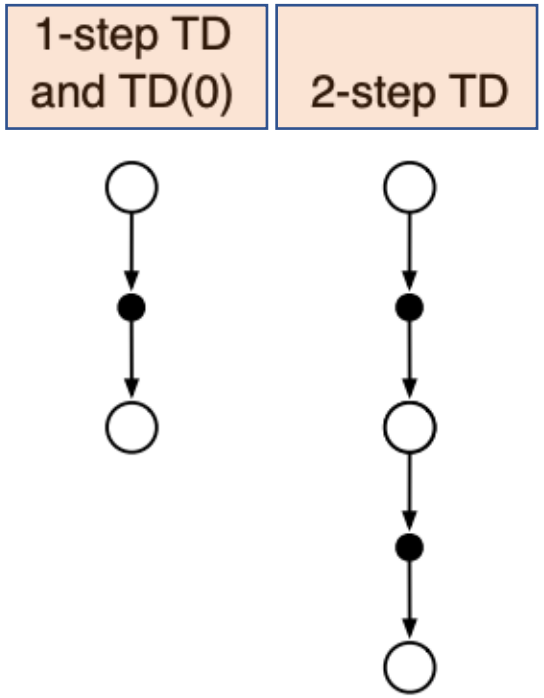


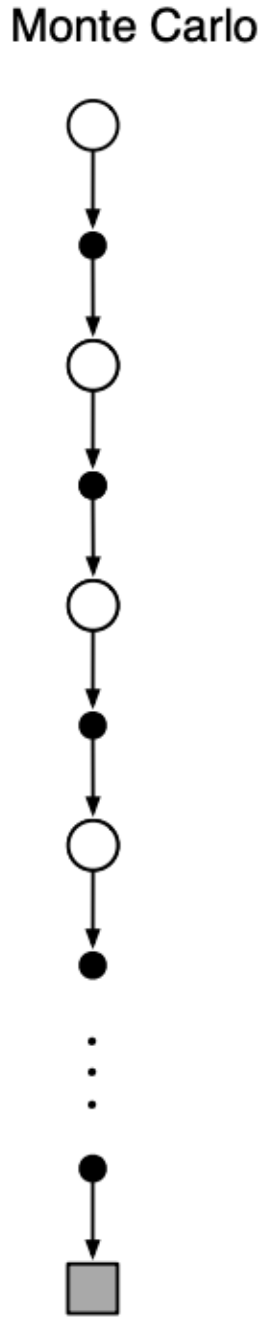
MC vs. TD(0)

MC: low bias

TD(0): low variance

Why don’t using a compromise between MC and TD(0)?





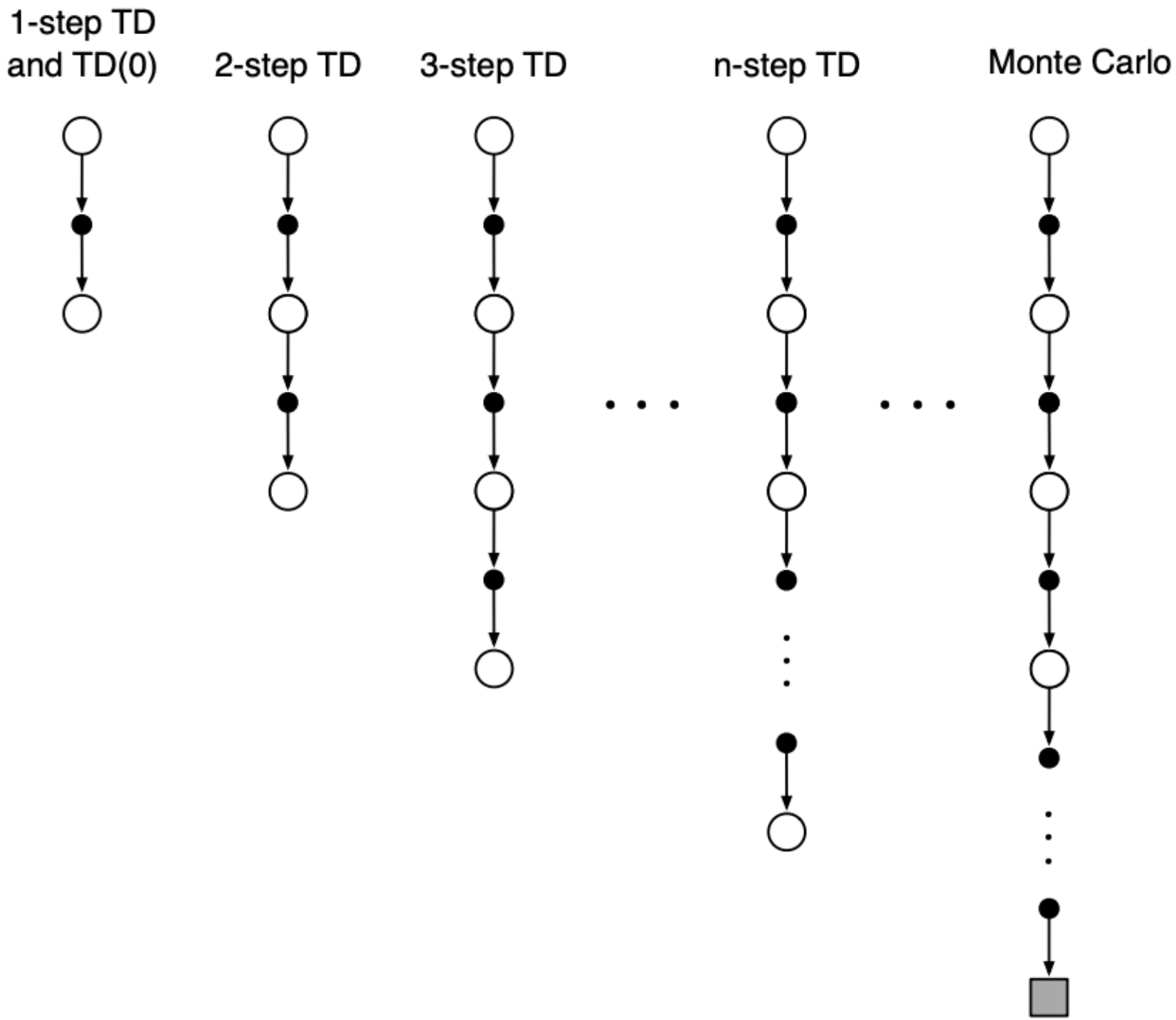
MC vs. TD(0)

MC: low bias

TD(0): low variance

Why don't using a compromise between MC and TD(0)?

We can generalize by considering n-step TD!



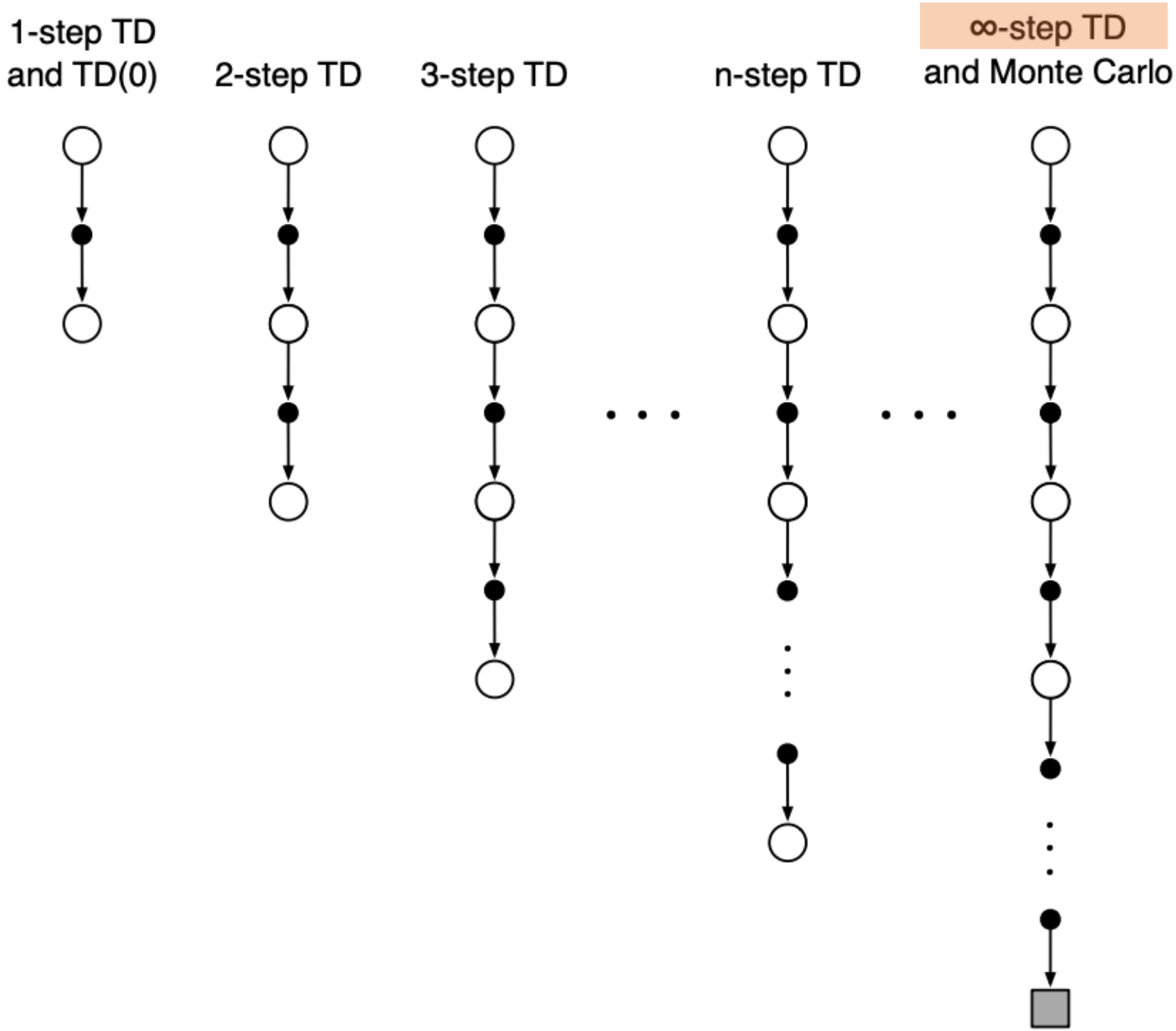
MC vs. TD(0)

MC: low bias

TD(0): low variance

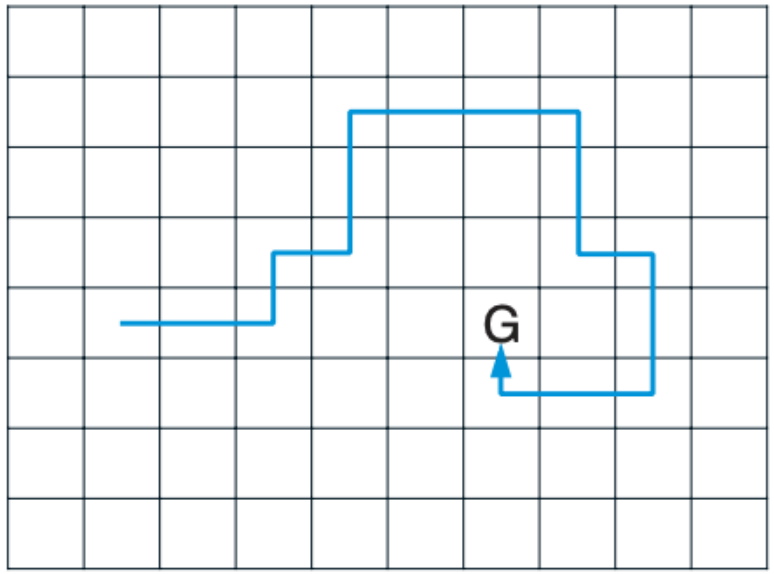
Why don't using a compromise between MC and TD(0)?

We can generalize by considering n-step TD!

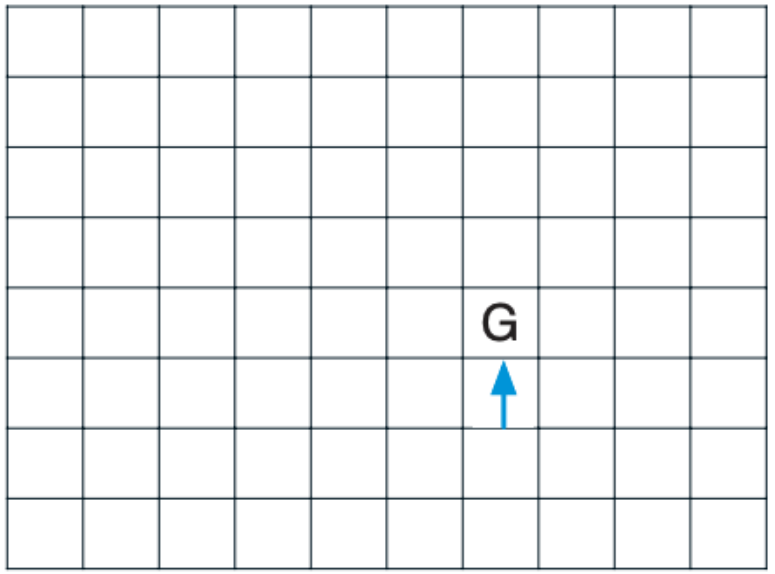


n-step TD learning (Prediction or Control)

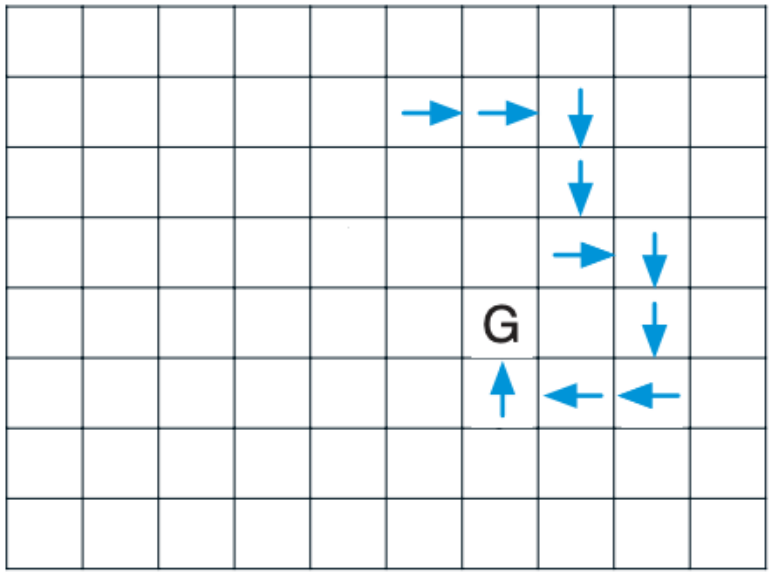
Path taken



Action values increased by one-step Sarsa



Action values increased by 10-step Sarsa



- The one-step method strengthens only the last action of the sequence of actions that led to the high reward
- The n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode.

Prediction: *n*-step return

Let's define the n-step returns:

$$
\begin{aligned}
n = 1 \quad (TD) \quad & G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}) \\
n = 2 \quad & G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \\
\vdots \quad & \vdots \\
n = \infty \quad (MC) \quad & G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T
\end{aligned}
$$

$$
G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})
$$

A terminal state

Prediction: *n*-step return

Let's define the n-step returns:

$$
\begin{aligned}
n = 1 \quad (TD) \quad & G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}) \\
n = 2 \quad & G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \\
\vdots \quad & \vdots \\
n = \infty \quad (MC) \quad & G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T
\end{aligned}
$$

$$
G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})
$$

n-step TD learning will consider the following updates:

$$
V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_{t:t+n} - V_{t+n-1}(S_t) \right]
$$

Prediction: *n*-step return

Where this is not defined?

Let's define the n-step returns:

$$
\begin{aligned}
n = 1 \quad (TD) \quad & G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}) \\
n = 2 \quad & G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \\
\vdots \quad & \vdots \\
n = \infty \quad (MC) \quad & G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T
\end{aligned}
$$

$$
G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})
$$

n-step TD learning will consider the following updates:

$$
V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_{t:t+n} - V_{t+n-1}(S_t) \right]
$$

Prediction: *n*-step return

Where this is not defined? When there are less then *n* step before the terminal state!

Let's define the n-step returns:

$$
\begin{aligned}
n = 1 \quad (TD) \quad & G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1}) \\
n = 2 \quad & G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2}) \\
\vdots \quad & \vdots \\
n = \infty \quad (MC) \quad & G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T
\end{aligned}
$$

$$
G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})
$$

n-step TD learning will consider the following updates:

$$
V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_{t:t+n} - V_{t+n-1}(S_t) \right]
$$

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

PREDICTION TASK

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\,[G - V(S_\tau)]$
  Until $\tau = T - 1$

A fixed value for n

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

t will be our step counter inside the episode

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positiv
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can

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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\,[G - V(S_\tau)]$
  Until $\tau = T - 1$

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

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positiv
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can

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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
      $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
  Until $\tau = T - 1$

t will be our step counter inside the episode
We will use:
- τ for starting the updates only when n steps are actually seen in the trajectory
- T for checking if we have arrived to close to a terminal state (and therefore a different update must be given)

We need to have at least n steps to do our updates: if we are too close to a terminal state we can't update anymore

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

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positiv
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can

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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
      $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
  Until $\tau = T - 1$

t will be our step counter inside the episode
We will use:
- τ for starting the updates only when n steps are actually seen in the trajectory
- T for checking if we have arrived to close to a terminal state (and therefore a different update must be given)

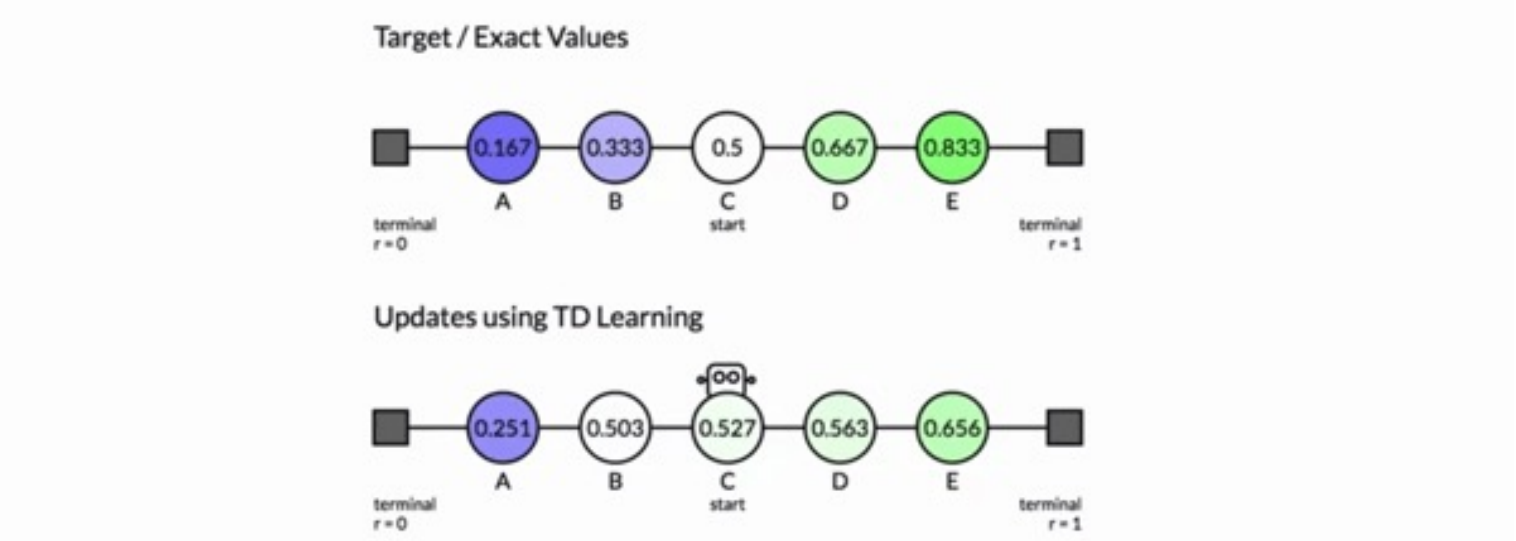
We need to have at least n steps to do our updates: if we are too close to a terminal state we can't update anymore

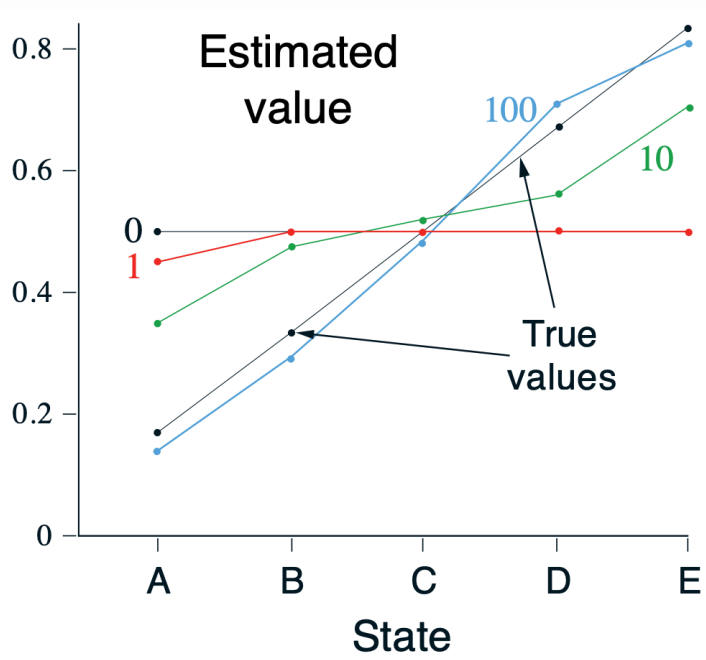
Policy-evaluation

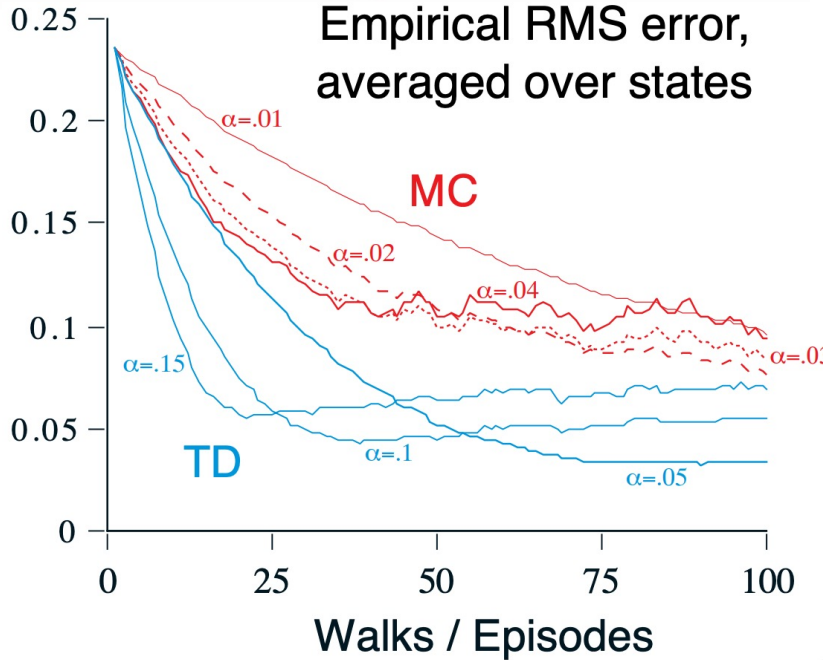
Prediction: *n*-step return – example

What is the ‘best’ n? It depends!

Let’s consider the random walk process example (we are evaluating a random uniform policy): instead of 5 states, we consider 19 states!



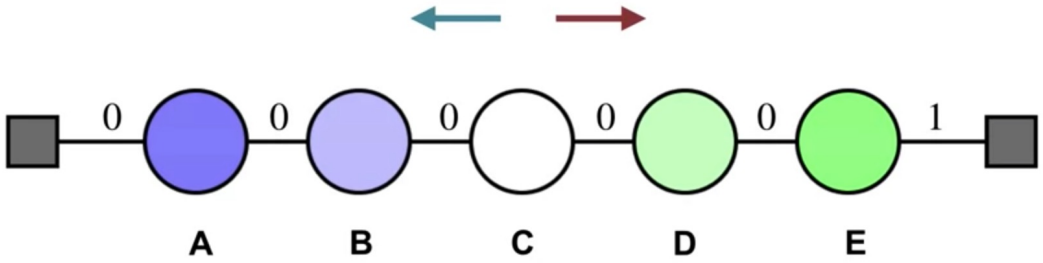




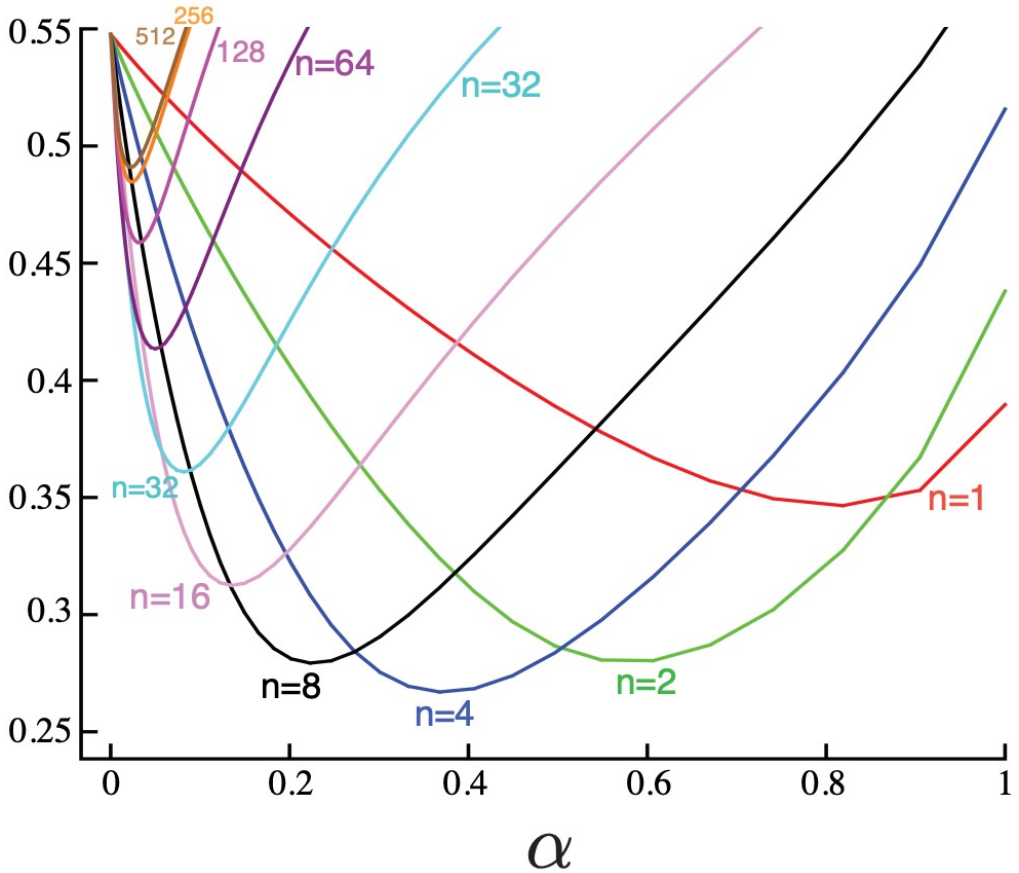
Prediction: *n*-step return – example

What is the ‘best’ n? It depends!

Let’s consider the random walk process example (we are evaluating a random uniform policy): instead of 5 states, we consider 19 states!



Average RMS error over 19 states and first 10 episodes



Control: *n*-step SARSA

Let's define the n-step returns:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} Q_{t+n-1}(S_{t+n}, A_{t+n})$$

n-step SARSA updates:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right]$$

n-step Sarsa for estimating $Q \approx q_*$ or q_π

CONTROL TASK

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_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:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t-n+1$ (τ is the time whose 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T-1$

A fixed value for n

Loop for each episode:

Initialize and store $S_0 \neq$ terminal

Select and store an action $A_0 \sim \pi(\cdot|S_0)$

$T \leftarrow \infty$

Loop for $t = 0, 1, 2, \ldots$:

If $t < T$, then:

Take action A_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:

$T \leftarrow t + 1$

else:

Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$

$\tau \leftarrow t - n + 1$ (τ is the time whose 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$

$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$

If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q

Until $\tau = T - 1$

t will be our step counter inside the episode

We will use:

- τ for starting the updates only when n steps are actually seen in the trajectory
- T for checking if we have arrived to close to a terminal state (and therefore a different update must be given)

Loop for each episode:

Initialize and store $S_0 \neq$ terminal

Select and store an action $A_0 \sim \pi(\cdot|S_0)$

$T \leftarrow \infty$

Loop for $t = 0, 1, 2, \ldots$:

| If $t < T$, then:

| | Take action A_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:

| | | $T \leftarrow t + 1$

| | else:

| | | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$

| $\tau \leftarrow t - n + 1$ (τ is the time whose 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$

| | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$

| | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q

Until $\tau = T - 1$

Policy-evaluation

Policy-improvement

Control: off-policy *n*-step SARSA

We can also have off-policy *n*-step, SARSA. We need to consider:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t) \right]$$

where an importance sampling ratio is in place

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h, T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim b(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | | Take action A_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:
    | | | $T \leftarrow t+1$
    | | else:
    | | | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    | If $\tau \geq 0$:
    | | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ ($\rho_{\tau+1:\tau+n}$)
    | | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
    | | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ ($G_{\tau:\tau+n}$)
    | | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
    | | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
    Until $\tau = T - 1$

Loop for each episode:

Initialize and store $S_0 \neq$ terminal

Select and store an action $A_0 \sim b(\cdot|S_0)$

Actions are taken according to a behaviour policy

$T \leftarrow \infty$

Loop for $t = 0, 1, 2, \ldots$:

| If $t < T$, then:

| | Take action A_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:

| | | $T \leftarrow t + 1$

| | else:

| | | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$

| $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)

| If $\tau \geq 0$:

| | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:\tau+n})$

| | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$

Target policy

| | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$

| | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\,[G - Q(S_\tau, A_\tau)]$

| | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q

Until $\tau = T - 1$

Loop for each episode:

Initialize and store $S_0 \neq$ terminal

Select and store an action $A_0 \sim b(\cdot|S_0)$

$T \leftarrow \infty$

Loop for $t = 0, 1, 2, \ldots$:

| If $t < T$, then:

| | Take action A_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:

| | | $T \leftarrow t + 1$

| | else:

| | | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$

| $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)

| If $\tau \geq 0$:

| | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ ($\rho_{\tau+1:\tau+n}$)

| | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$

| | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ ($G_{\tau:\tau+n}$)

| | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\,[G - Q(S_\tau, A_\tau)]$

| | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q

Until $\tau = T - 1$

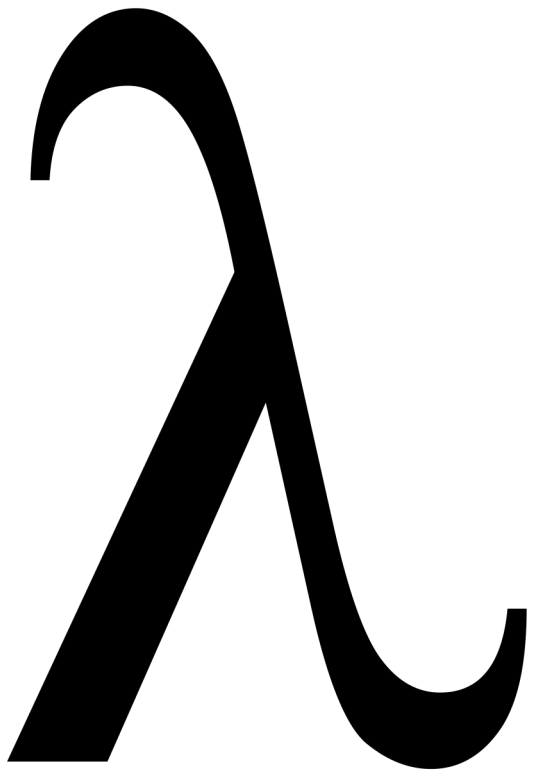
Importance Sampling must be considered

Target policy

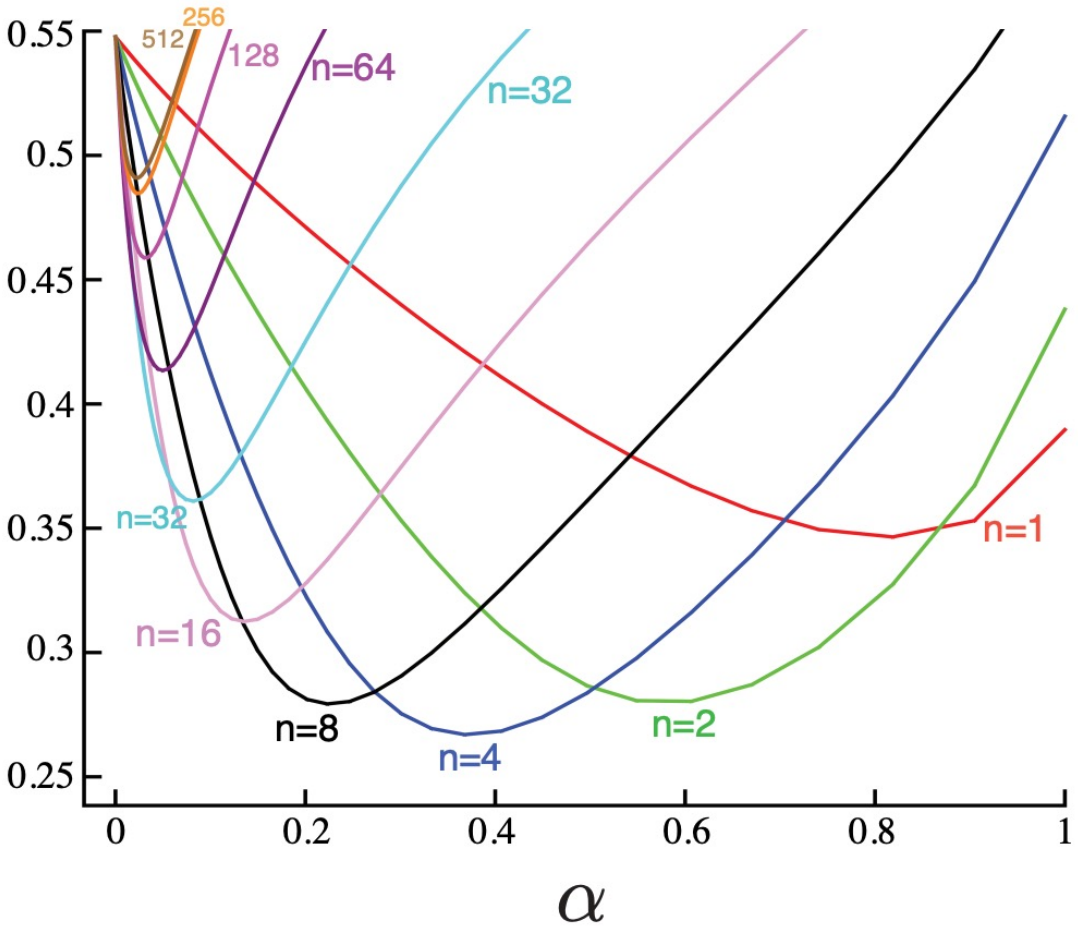
n-step bootstrap: Exam

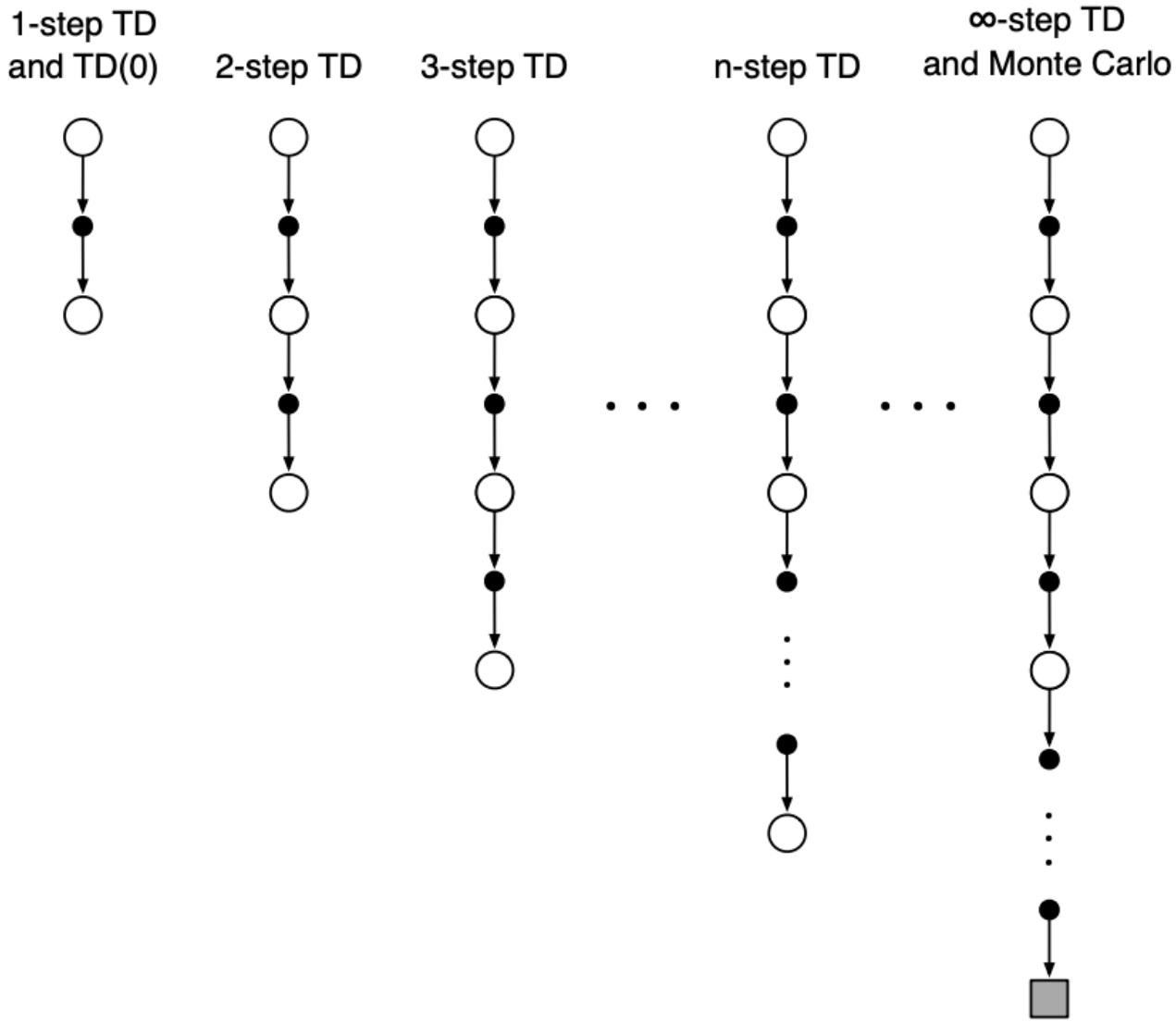
- Section 7.1 and 7.2 are exam material
- Section 7.3 (off-policy SARSA) is optional
- (Again!) Pay particular attention to the algorithms!

TD(λ)
(Chapter 12)

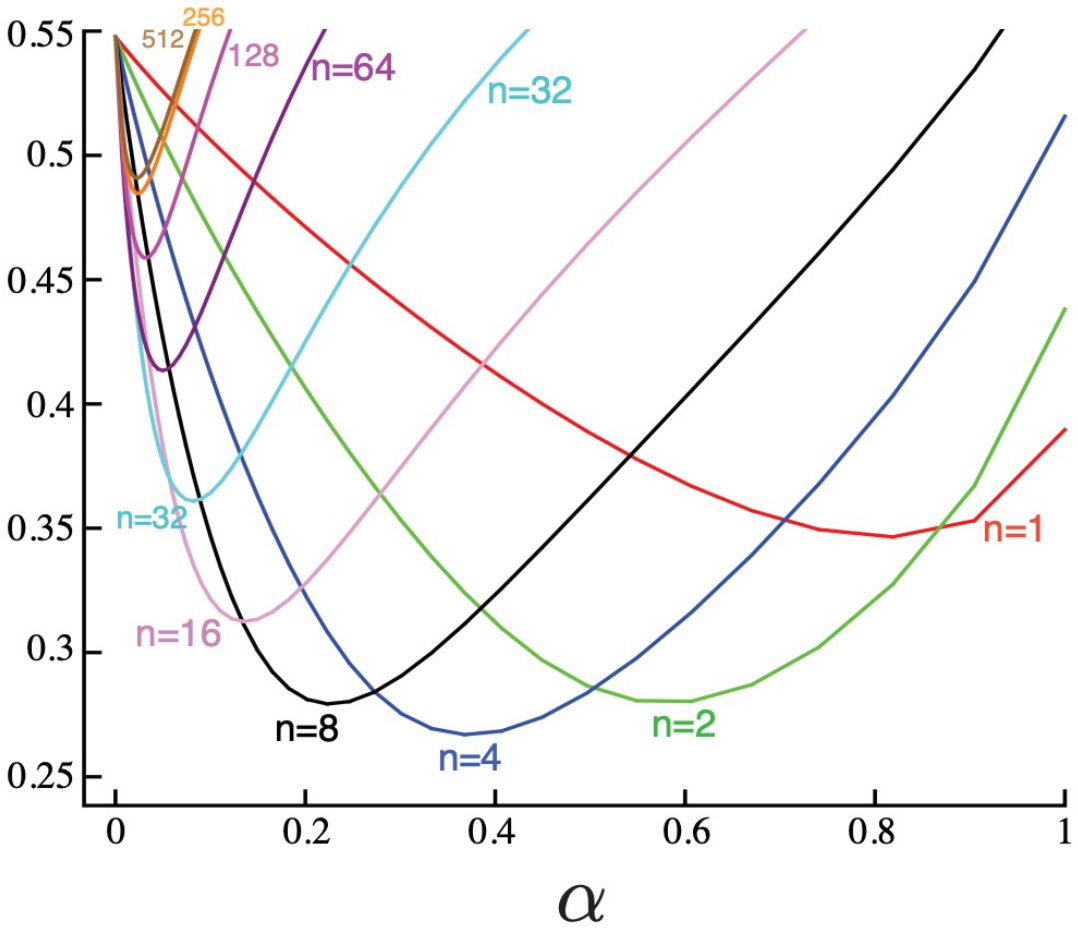


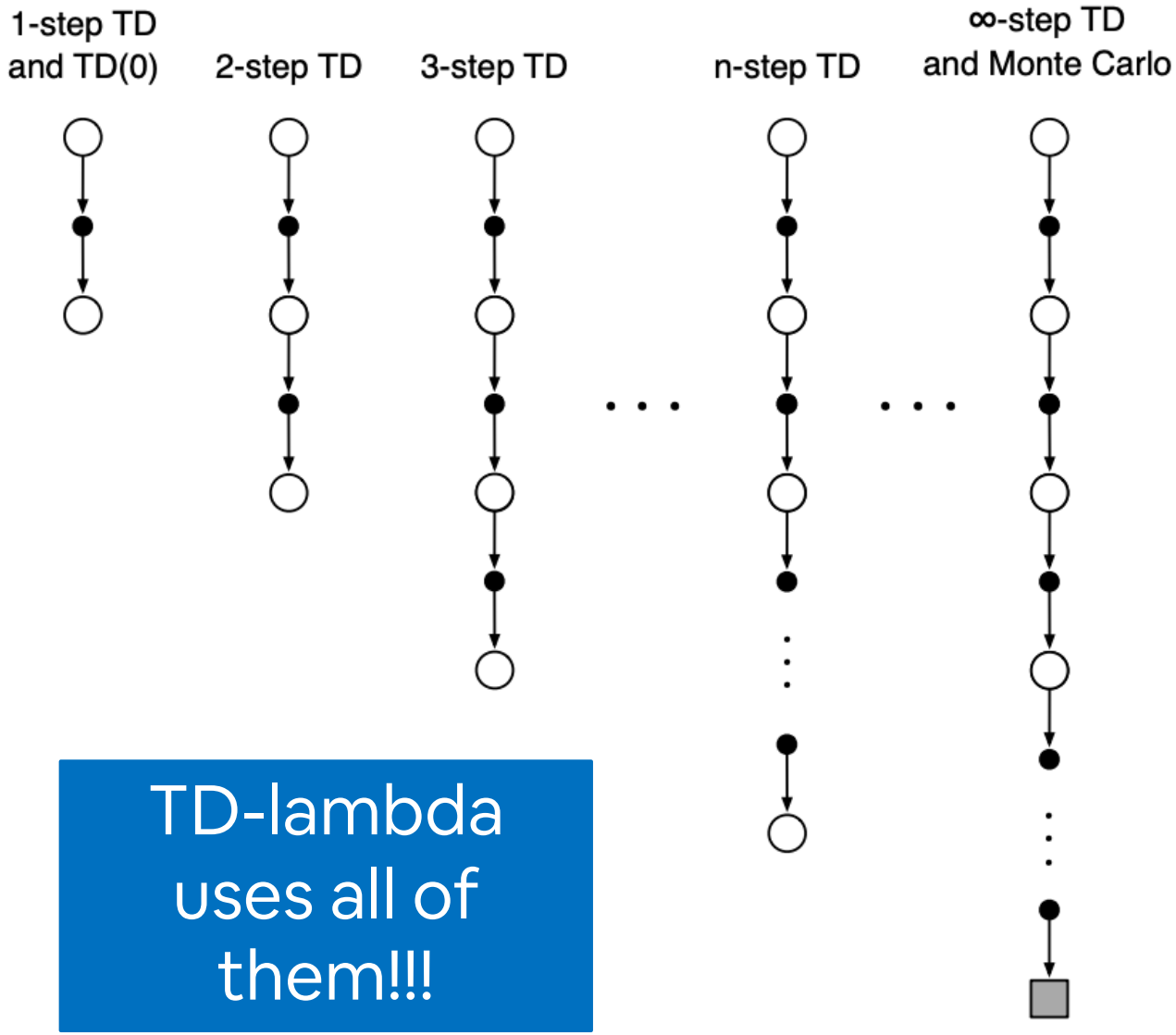
n-step: which *n*?





n-step: which *n*?

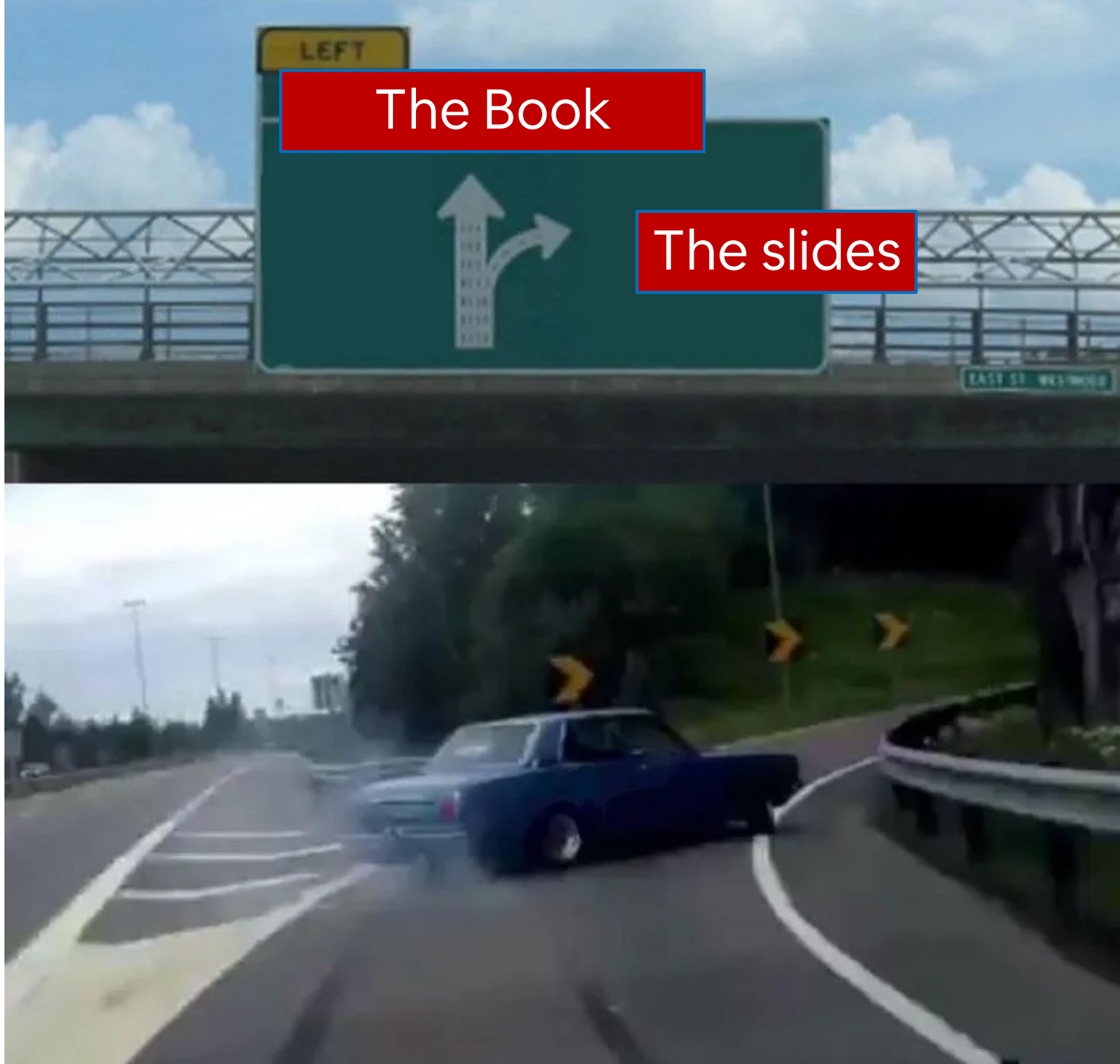




TD-lambda uses all of them!!!

Chapter 12

- The last version of the book described TD-lambda and *eligibility traces* (one of the main ‘tool’ in TD-lambda) in Chapter 12…
- … but it is mixed with value function approximation VFA algorithms (in my opinion it is a bit confusing)
- For the moment, slides are the reference, after we do VFA (starting next week) the TD-lambda algorithms reported in the book will be clear

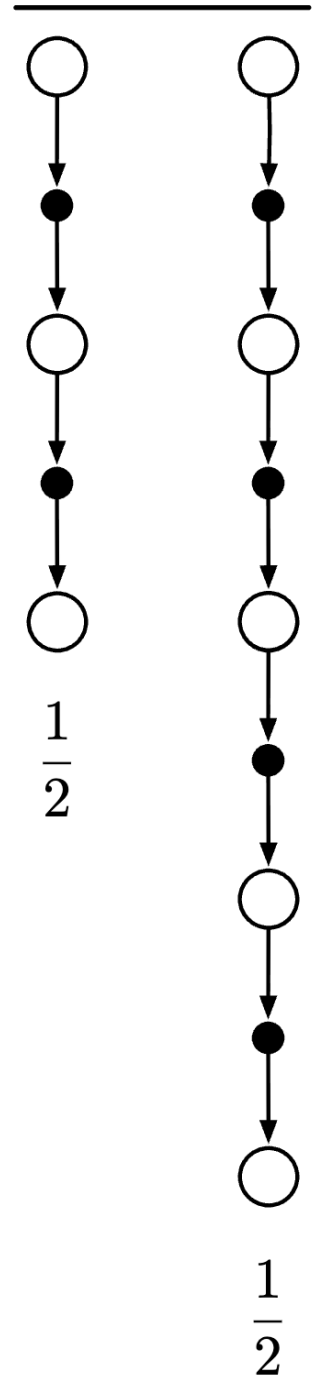


Averaging n-Step Returns

- We can average n-step returns over different n
- e.g. average the 2-step and 4-step returns
- Combines information from two different time-steps

$$\frac{1}{2}G_{t:t+2} + \frac{1}{2}G_{t:t+4}$$

- Can we efficiently combine information from all time-steps?



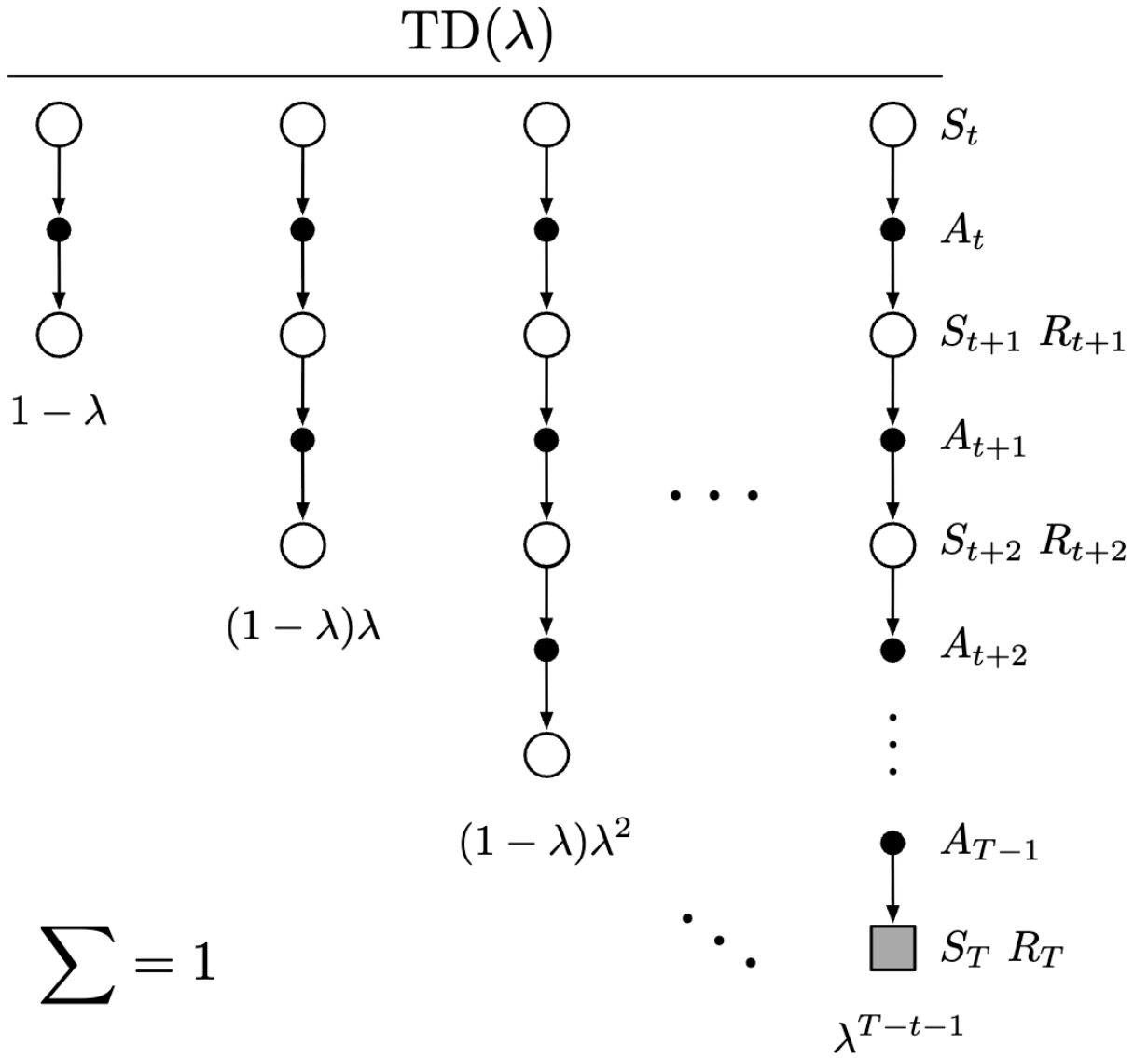
λ – return

- We consider the λ – return G_t^{λ} that combines all n-step returns
- It is a weighted sum of all the returns

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

- Instead of the 'usual' return we update

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\lambda} - V(S_t)\right)$$



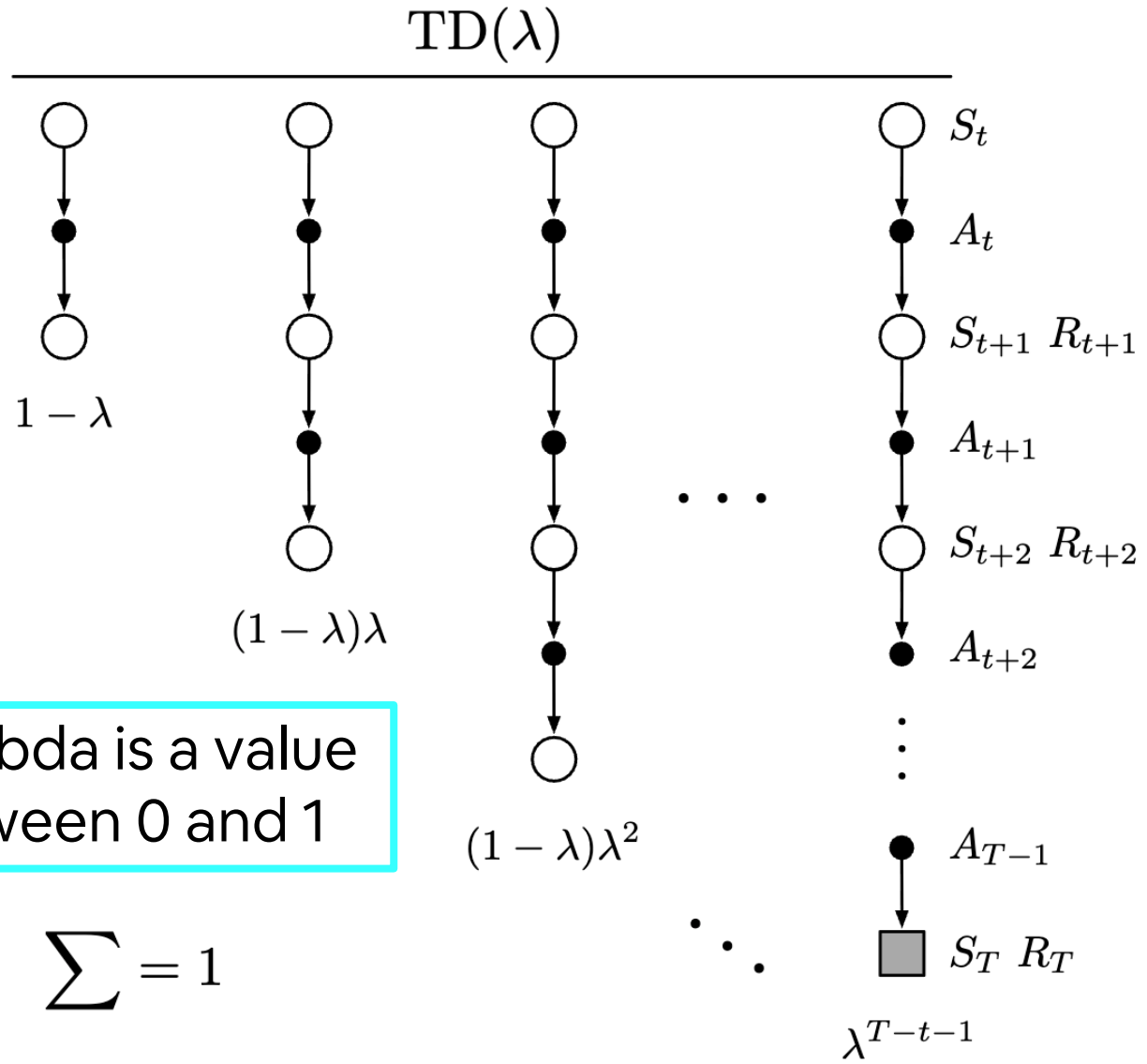
λ – return

- We consider the λ – return G_t^λ that combines all n-step returns
- It is a weighted sum of all the returns

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

- Instead of the 'usual' return we update

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

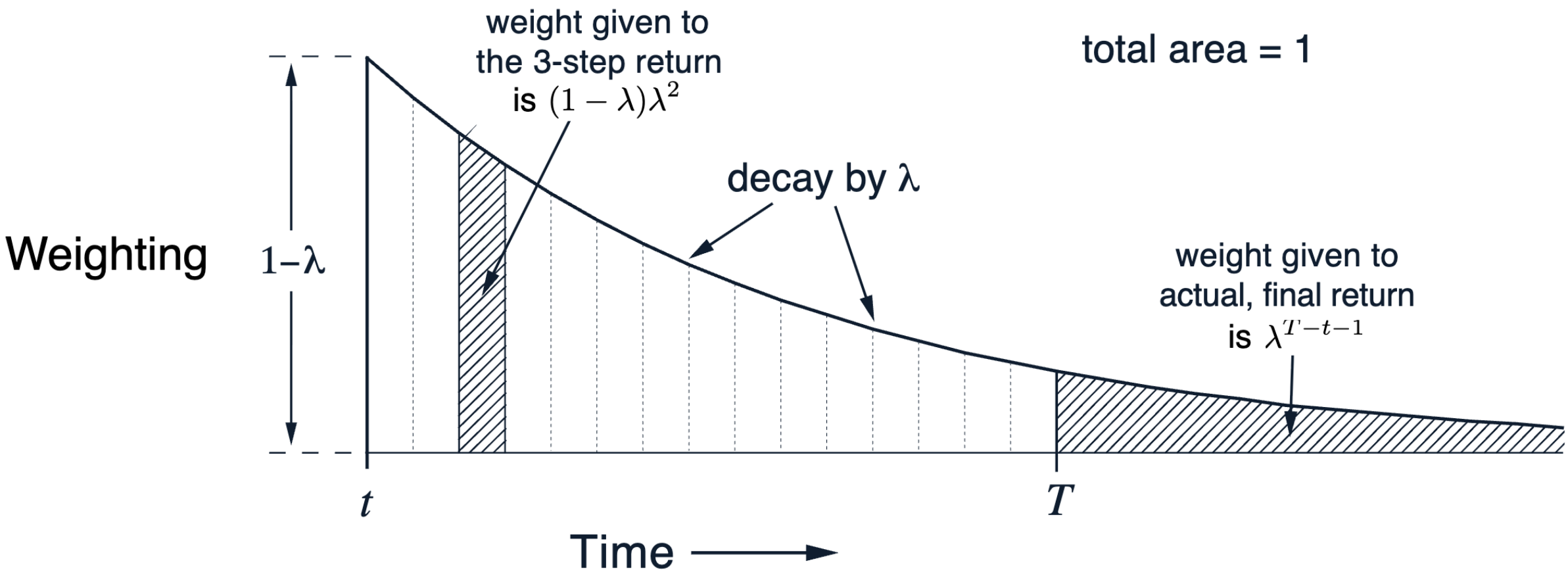


Lambda is a value between 0 and 1

$$\sum = 1$$

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^n = \frac{1}{1-\lambda}$$

TD(λ) Weighting Function



Why TD(λ) Weighting Function?

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

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

<u>Why we weight the steps?</u>

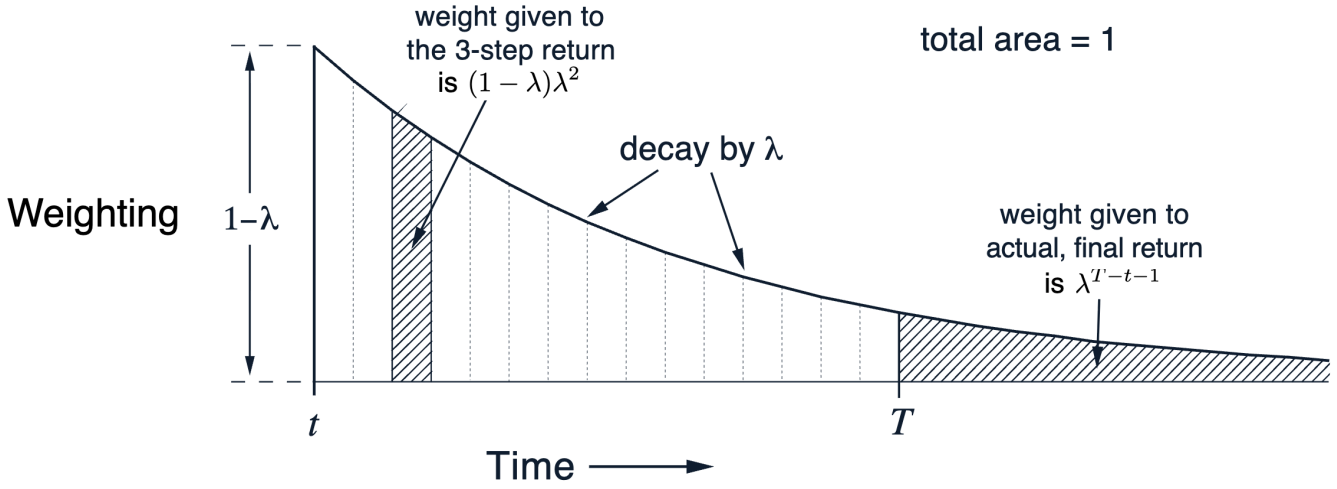
The weights form a geometric decay.

That means:

- 1-step return gets the highest weight

- 2-step, 3-step returns get smaller and smaller weights

This is intentional — it expresses how far into the future we trust our rollouts.



Why TD(λ) Weighting Function?

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

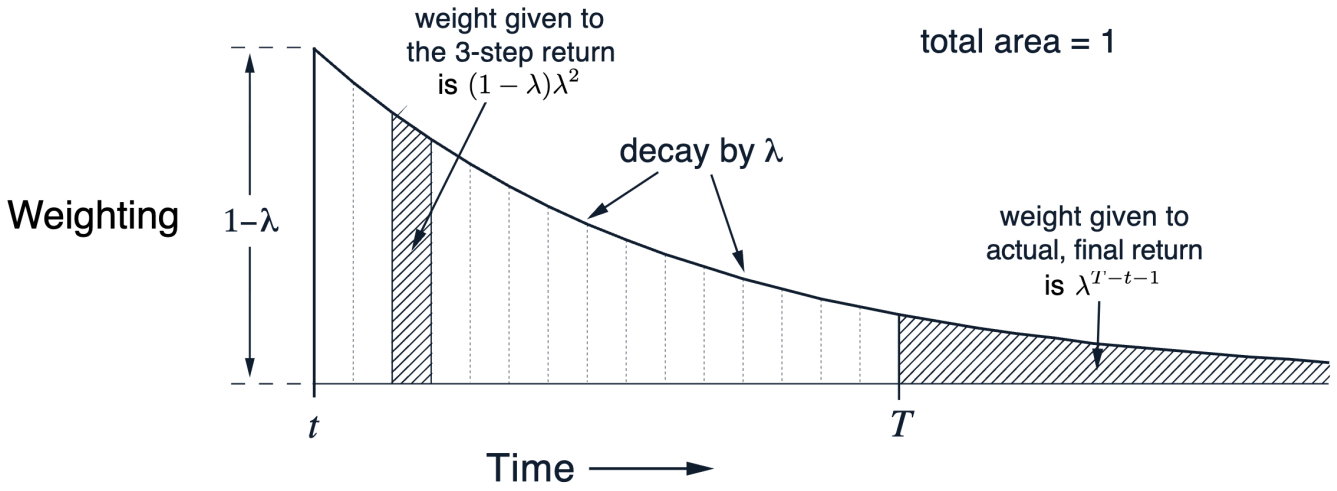
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

Why not give all steps equal weight?
With that idea:

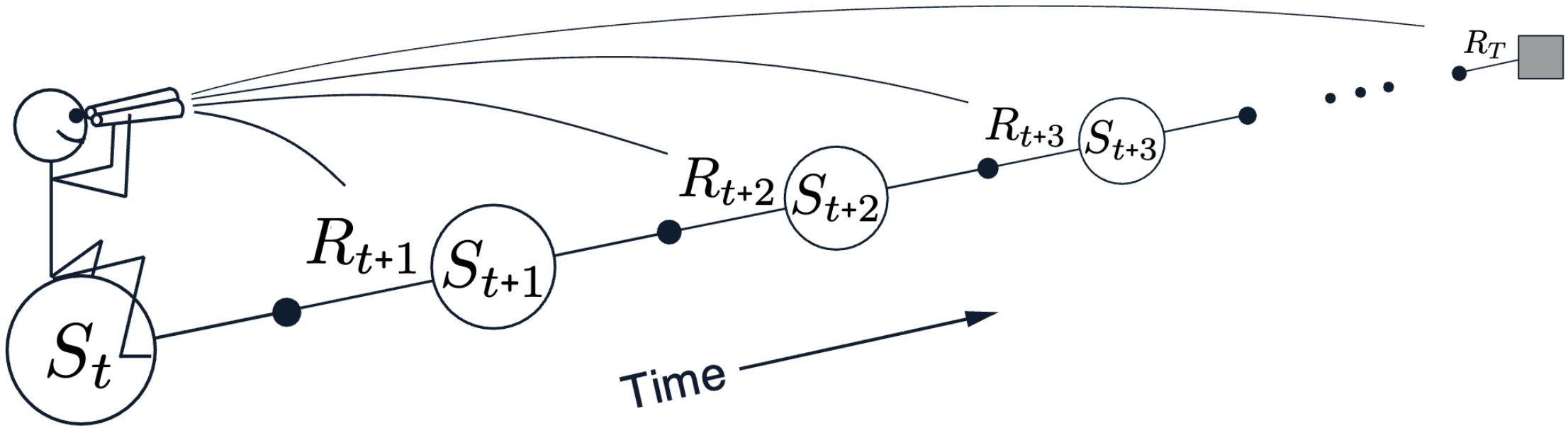
- You'd be mixing together very noisy long-term estimates with short-term, low-variance estimates.
- The result wouldn't correspond to any meaningful bias–variance trade-off.
- In practice, it would be unstable, especially when the environment is long or continuous.

The exponential decay in λ ensures that:

- recent TD errors matter more (because they're more reliable)
- but you still get some credit assignment from later steps.



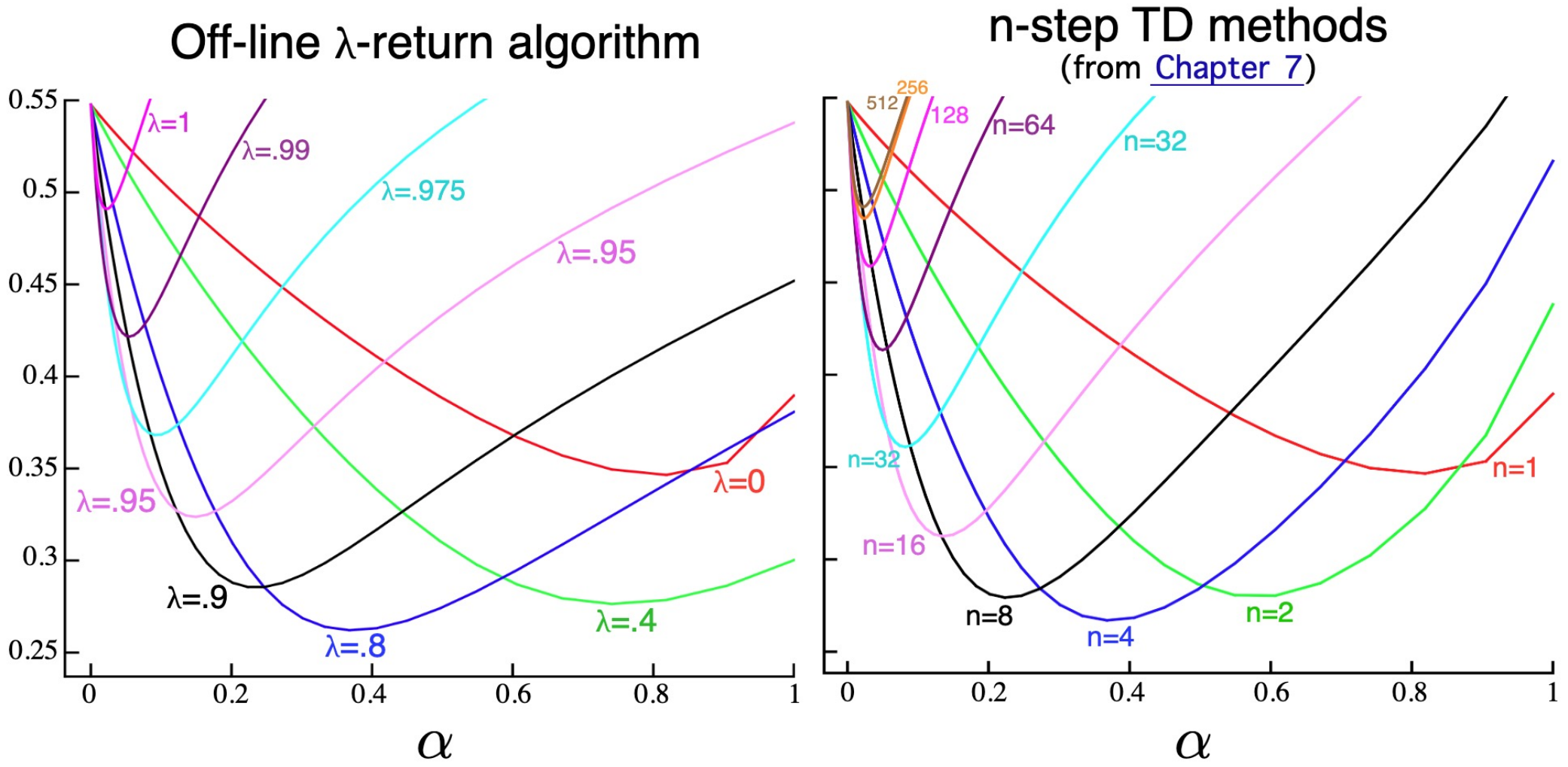
(Forward-view) TD(λ)



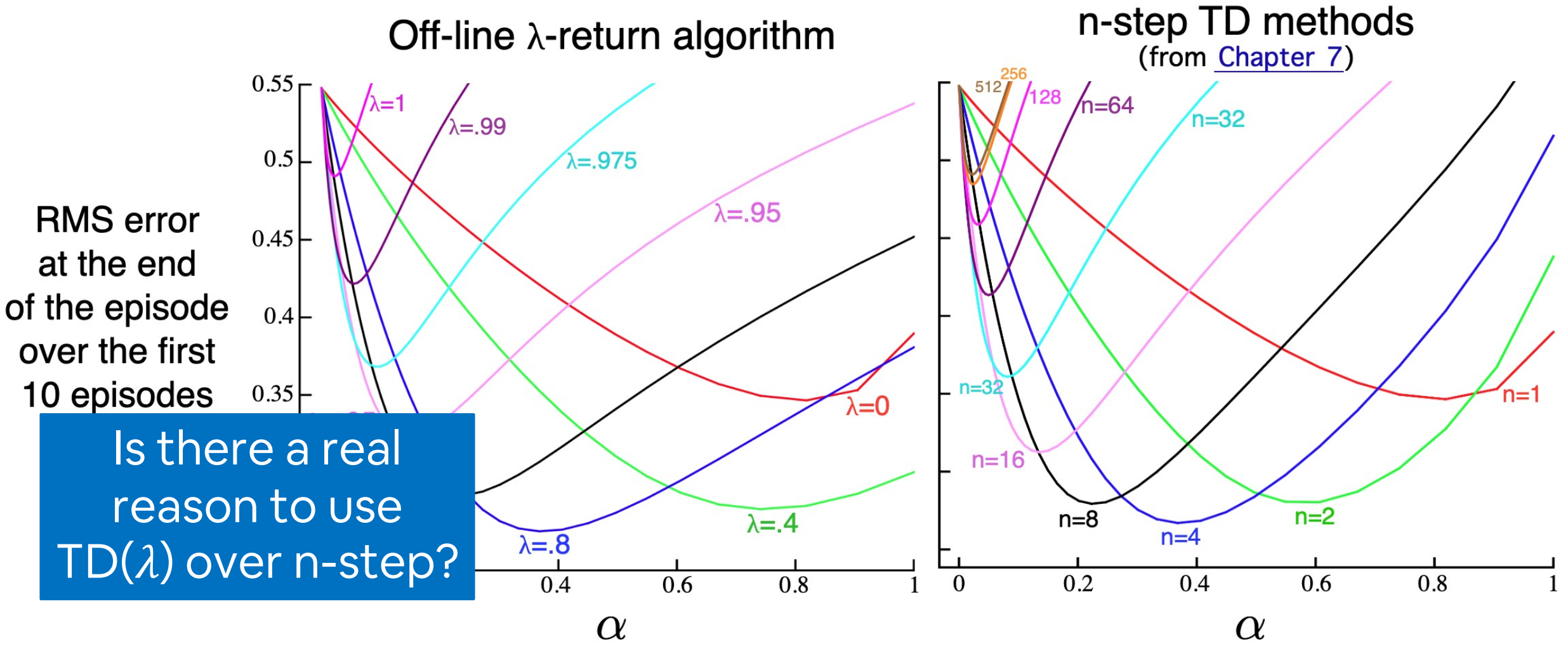
- Update value function towards the λ-return
- Forward-view looks into the future to compute λ – return G_t^{λ}
- Like Monte Carlo, it can only be computed from complete episodes

TD(λ) in action: ex. Prediction random walk

RMS error at the end of the episode over the first 10 episodes

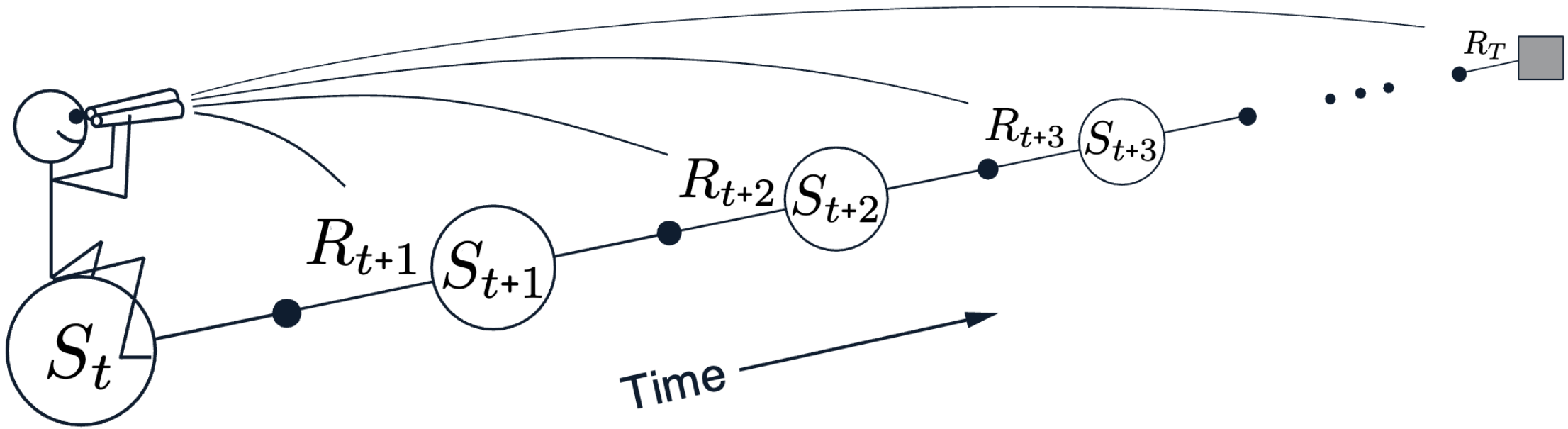


TD(λ) in action: ex. Prediction random walk



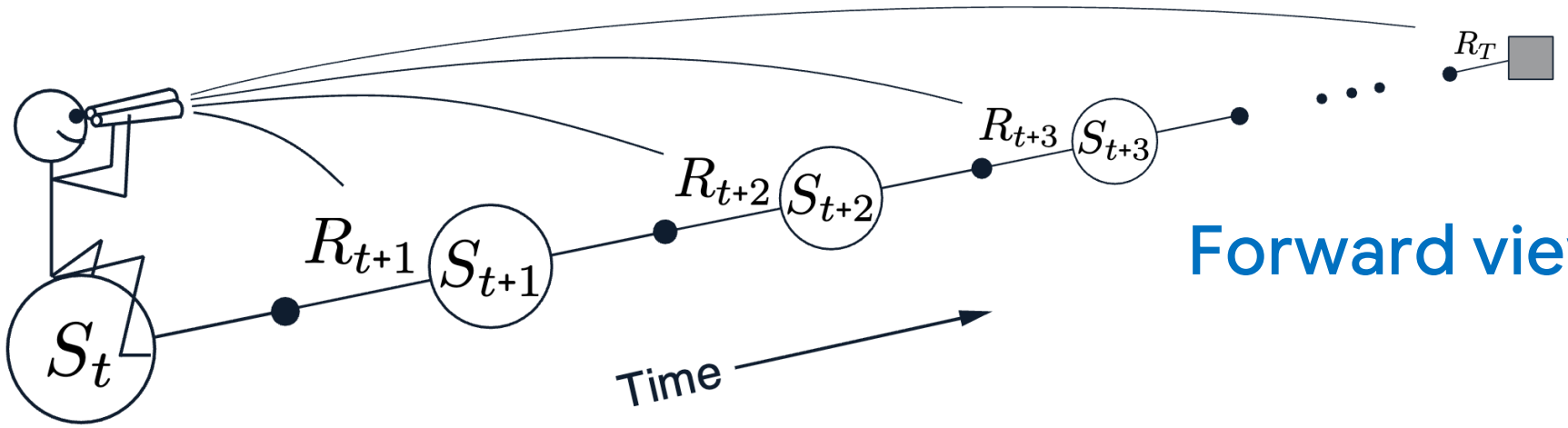
Is there a real reason to use TD(λ) over n-step?

Forward-view TD(λ)

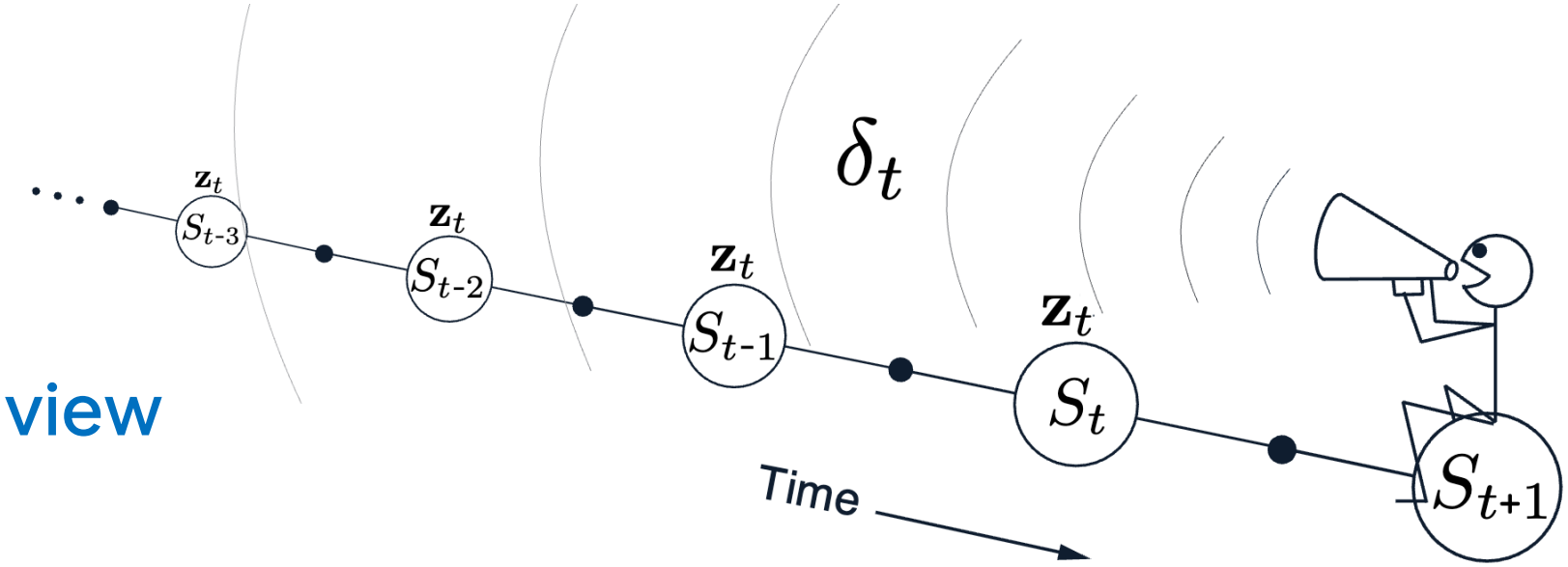


Forward views are somehow complex to implement because the update of each state depends on later events or rewards that are not available at current time

Let's change our perspective!

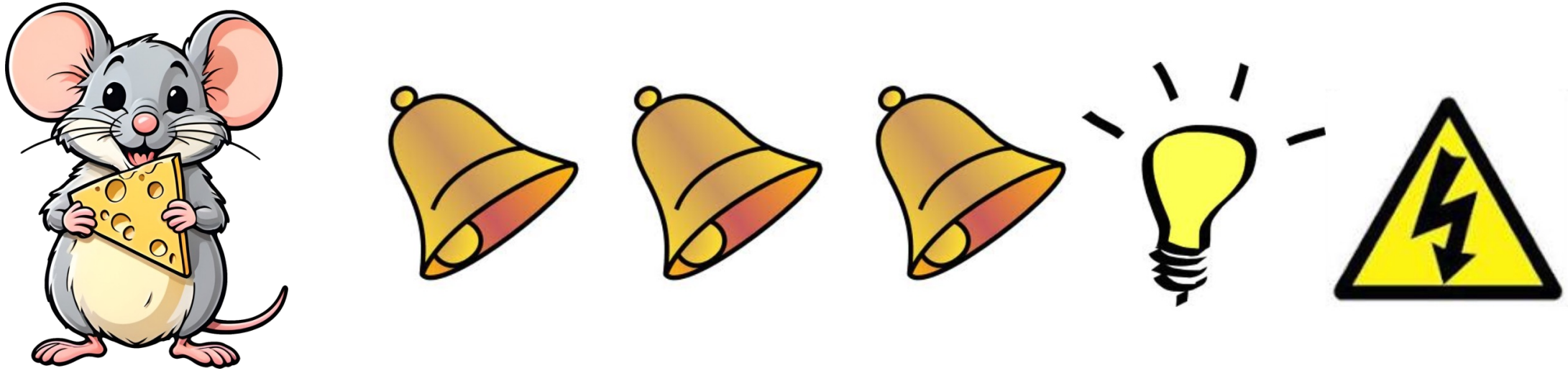


Forward view



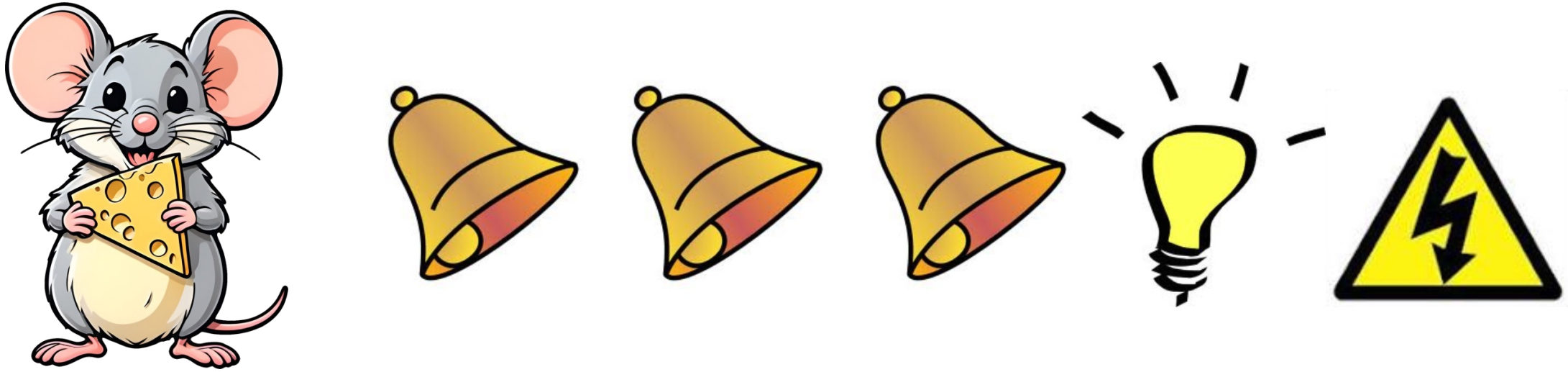
Backward view

Eligibility traces



- Credit assignment problem: did bell or light cause shock?

Eligibility traces

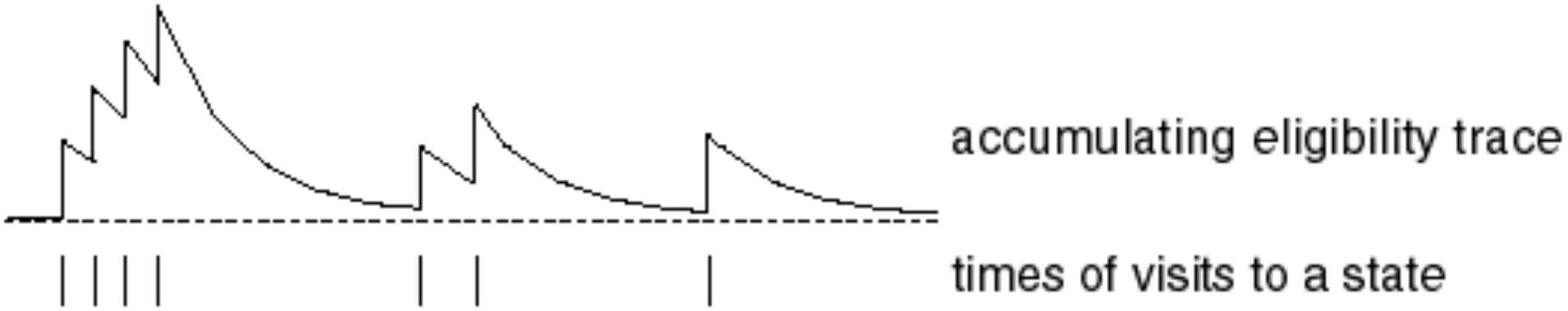


- Credit assignment problem: did bell or light cause shock?
- Frequency heuristic: assign credit to most frequent states
- Recency heuristic: assign credit to most recent states

Eligibility traces

$$E_0(s) = 0$$

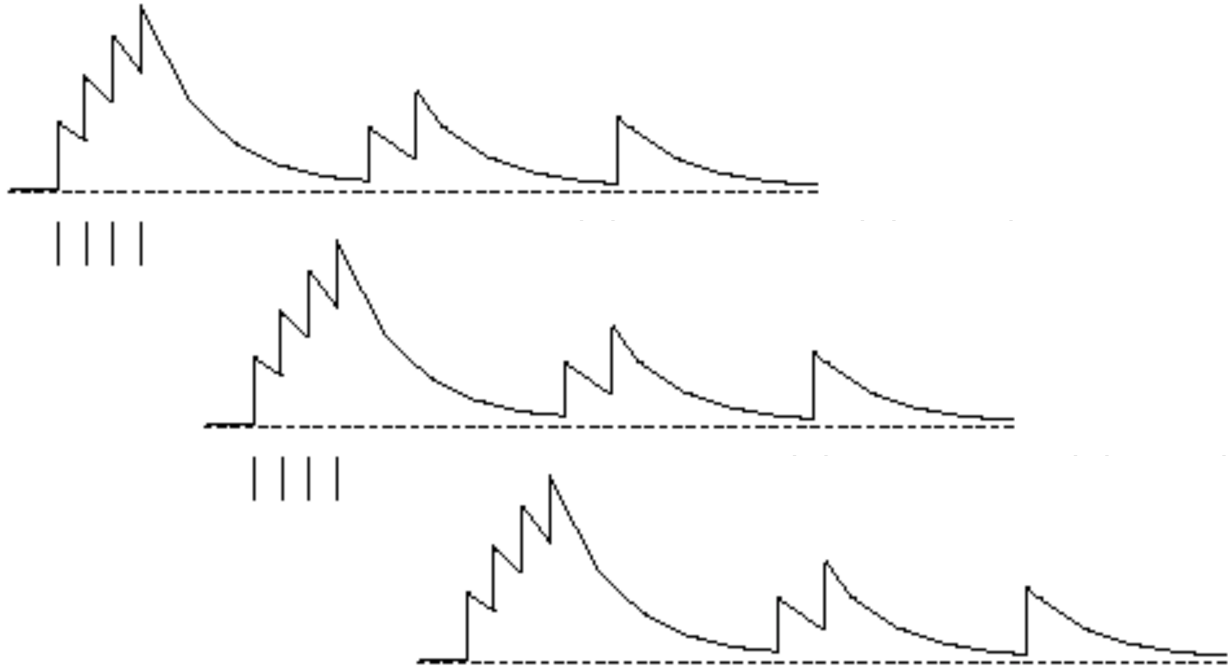
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$



I have on eligibility trace for every state in the MDP and they evolve over the episode!

Eligibility traces (prediction)

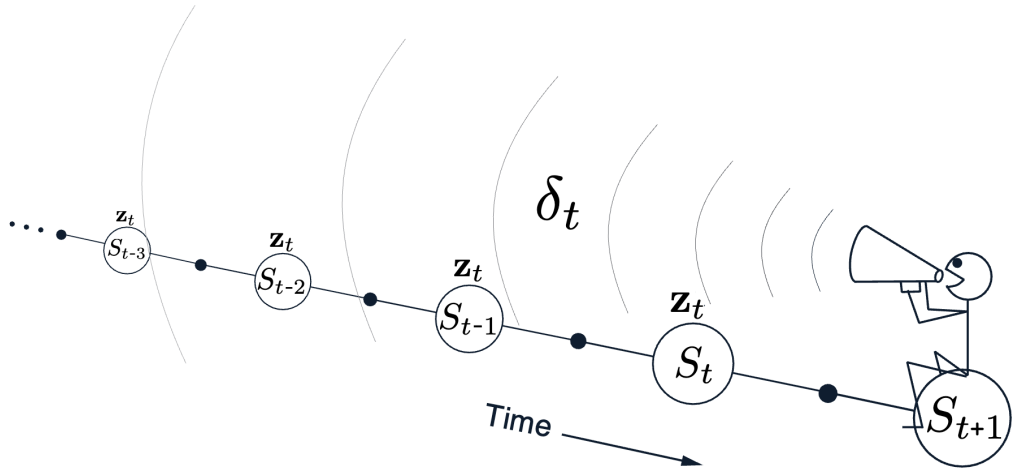
$$E_0(s) = 0$$
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$



We have an update for all s (!) at each step (like in TD-0) that depends on the TD-error and the eligibility traces

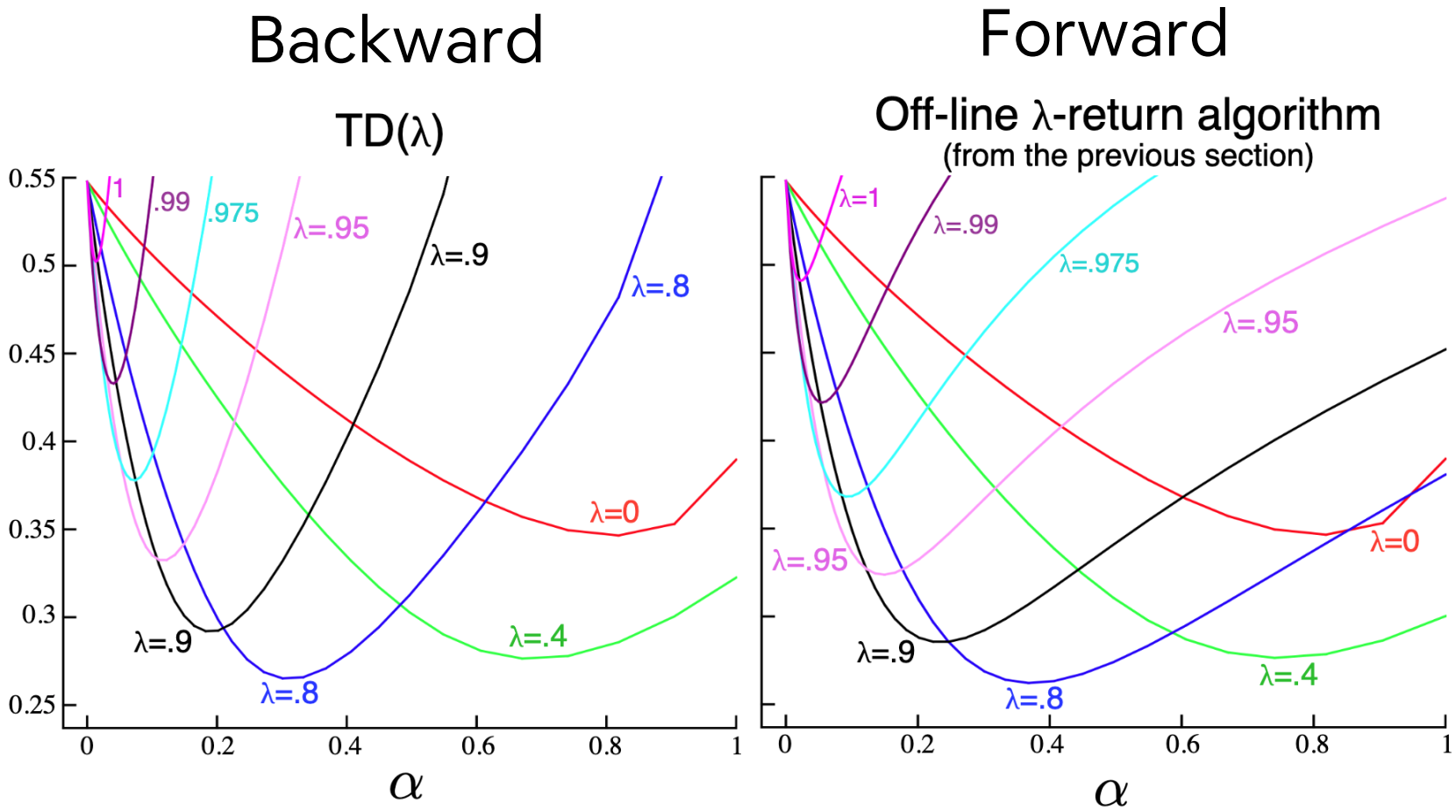
$$\delta_t \;=\; R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$$

$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$



Forward vs Backward view (prediction) - offline

RMS error
at the end
of the episode
over the first
10 episodes



Off-line (after we have completed the episode) we have equivalence of the two approaches... but backward view works without waiting the end of the episode!

Credits

- Image of the course is taken from C. Mahoney ‘Reinforcement Learning’ https://towardsdatascience.com/reinforcement-learning-fda8ff535bb6

Università degli Studi di Padova

Reinforcement Learning 2024/2025

RL2 Reinforcement Learning Research Lab @ UniPD

Thank you! Questions?

Gian Antonio Susto

Ruggero Carli