Temporal Difference Learning

with some slides stolen from David Silver

Previous lecture

- Dynamic Programming: first steps to solve an MDP

Previous lecture

- Dynamic Programming: first steps to solve an MDP
  - Problem: MDP needed to be known

Previous lecture

- Dynamic Programming: first steps to solve an MDP
  - Problem: MDP needed to be known
- Monte Carlo Learning

Previous lecture

- Dynamic Programming: first steps to solve an MDP
  - Problem: MDP needed to be known
- Monte Carlo Learning
  - Much better: MDP was allowed to be unknown

Previous lecture

- Dynamic Programming: first steps to solve an MDP
  - Problem: MDP needed to be known
- Monte Carlo Learning
  - Much better: MDP was allowed to be unknown
  - learned through sampling episodes

Previous lecture

- Dynamic Programming: first steps to solve an MDP
  - Problem: MDP needed to be known
- Monte Carlo Learning
  - Much better: MDP was allowed to be unknown
  - learned through sampling episodes
  - Started with learning State value function.

Previous lecture

- Dynamic Programming: first steps to solve an MDP
  - Problem: MDP needed to be known
- Monte Carlo Learning
  - Much better: MDP was allowed to be unknown
  - learned through sampling episodes
  - Started with learning State value function.
  - Problem for optimising! What did we do?

Previous lecture

- Dynamic Programming: first steps to solve an MDP
  - Problem: MDP needed to be known
- Monte Carlo Learning
  - Much better: MDP was allowed to be unknown
  - learned through sampling episodes
  - Started with learning State value function.
  - Problem for optimising! What did we do?
  - Problem,: episodic environments needed: Episodes must always end. No way to update online. Reason?

Previous lecture

- Dynamic Programming: first steps to solve an MDP
  - Problem: MDP needed to be known
- Monte Carlo Learning
  - Much better: MDP was allowed to be unknown
  - learned through sampling episodes
  - Started with learning State value function.
  - Problem for optimising! What did we do?
  - Problem,: episodic environments needed: Episodes must always end. No way to update online. Reason?
  - Episodes were also short in the black-jack example.

Temporal Difference Learning

- TD methods learn directly from episodes of experience
- TD is model-free
- TD learns from *incomplete* episodes, by bootstrapping
- TD updates a guess towards a guessed return
- TD(λ) is able to improve the guess towards the guessed return: Long-term lookout
-

Temporal difference learning

Incremental Monte-Carlo Updates

- Goal: learn v_π online demo experiences under the policy π
  - Incrementally with every new episode
- Update each $V(s)$ after each episode E^i towards the ***actual return*** G_t:
  - For every state S_t with return G_t
    - increment counter $N(s) \leftarrow N(s) + 1$
    - estimated mean return $V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$

TD learning

- Goal: learn v_π online demo experiences under the policy π

- Simplest temporal difference learning algorithm: TD(0)
  - Update $V(S_t)$ towards the ***estimated return*** $R_{t+1} + \gamma V(S_{t+1})$

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

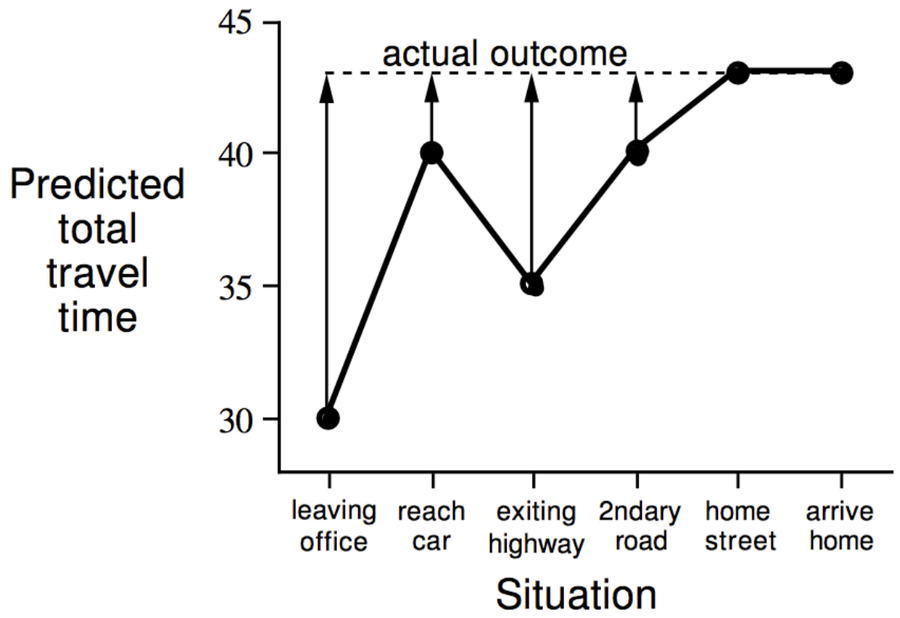
  - $R_{t+1} + \gamma V(S_{t+1})$ is called *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called *TD error*

driving home: Update towards the best guess

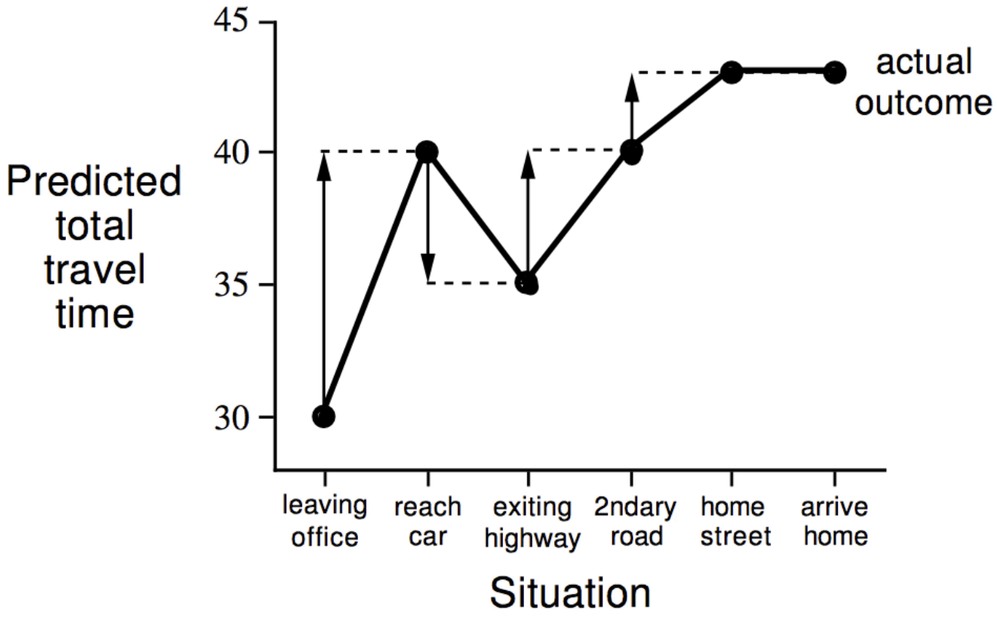
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

driving home: Update towards the best guess

Changes recommended by Monte Carlo methods (α=1)



Changes recommended by TD methods (α=1)



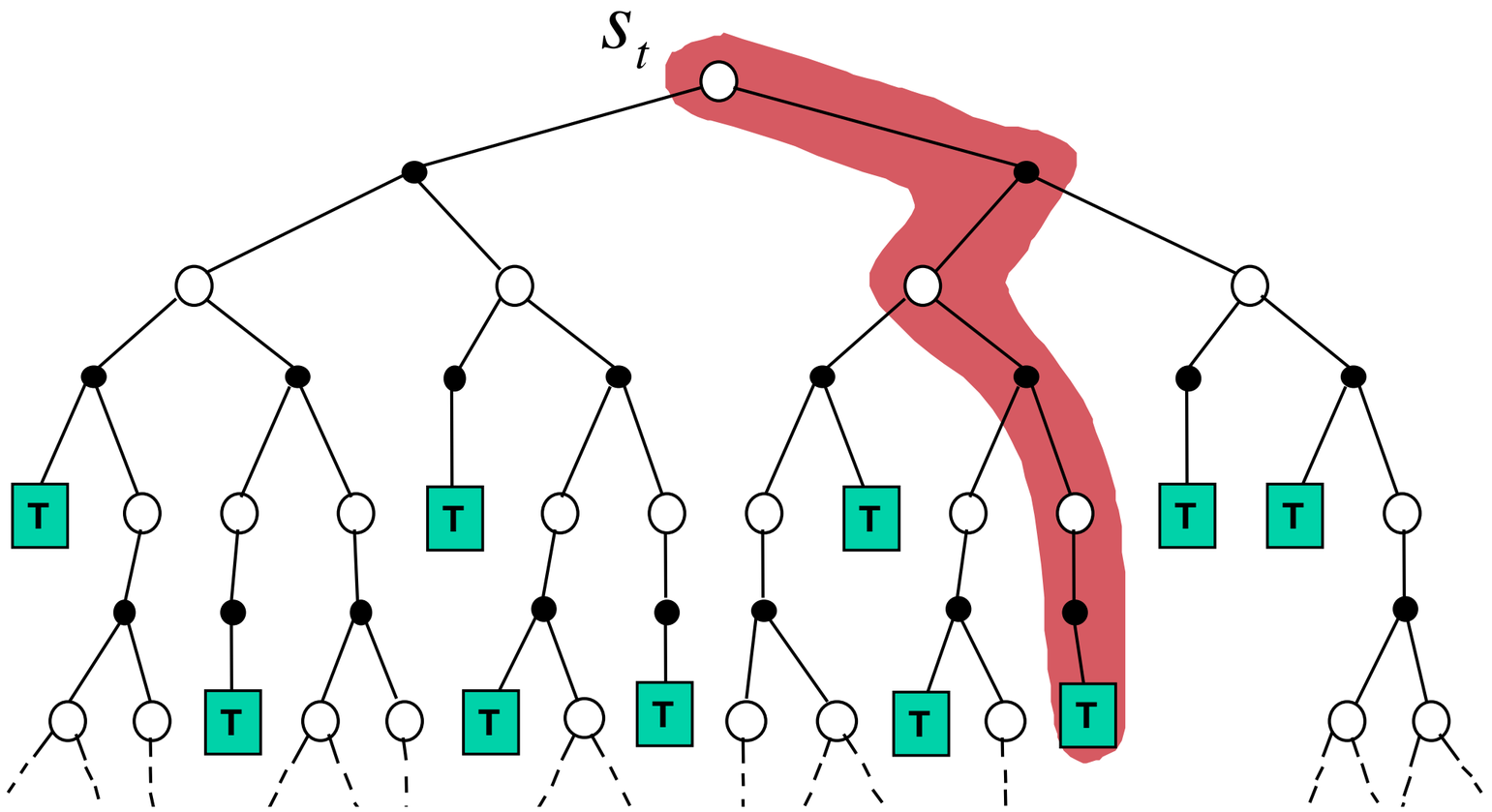
LUND UNIVERSITY

Comparison TD vs MC

- TD: updates *online towards the best guess*, *before* the final outcome
- MC has to wait until the episode is complete and we know the final return G_t
- TD learns from incomplete episodes, e.g., without final outcome.
- MC requires complete episodes

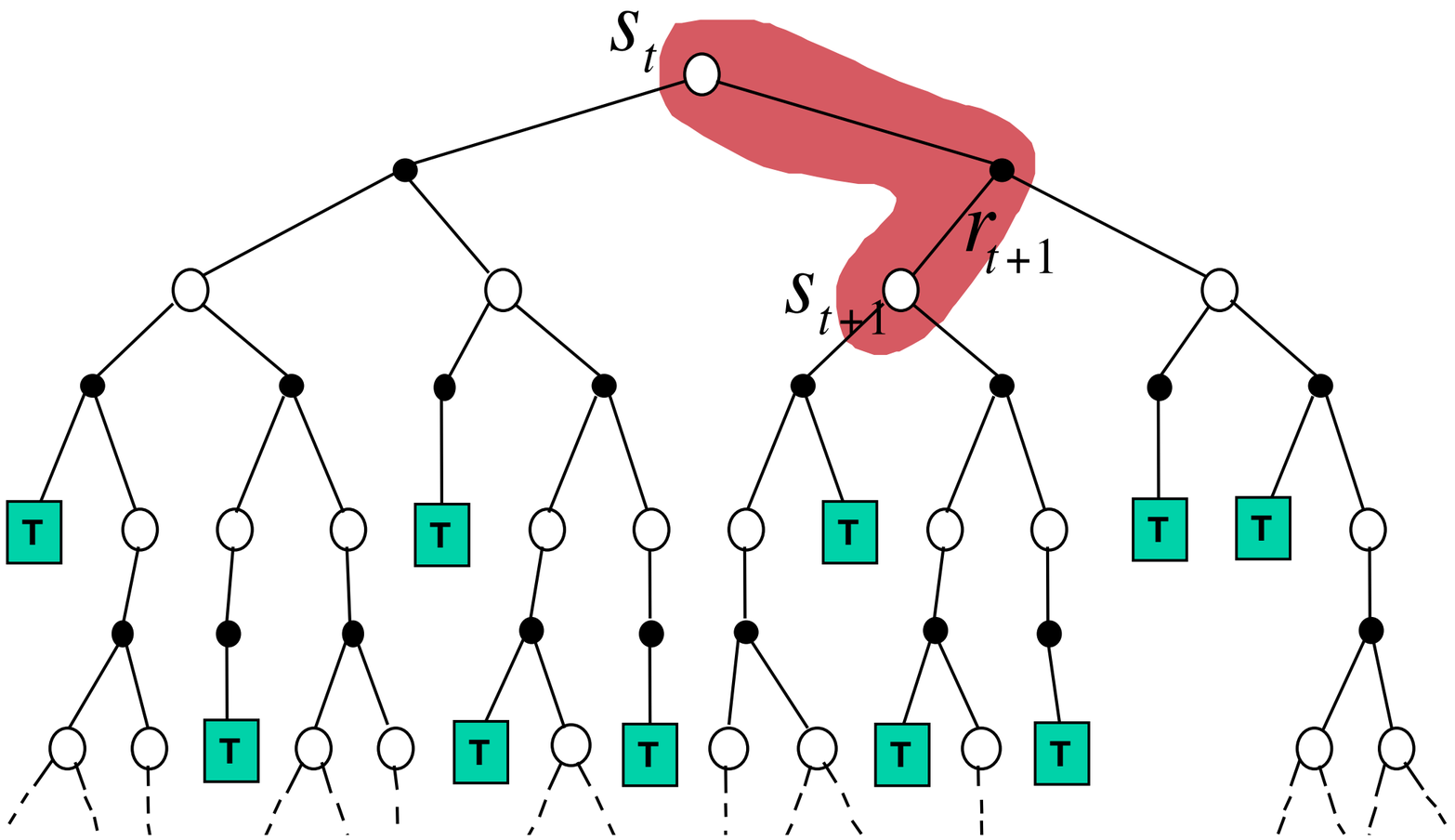
Monte Carlo Backup

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



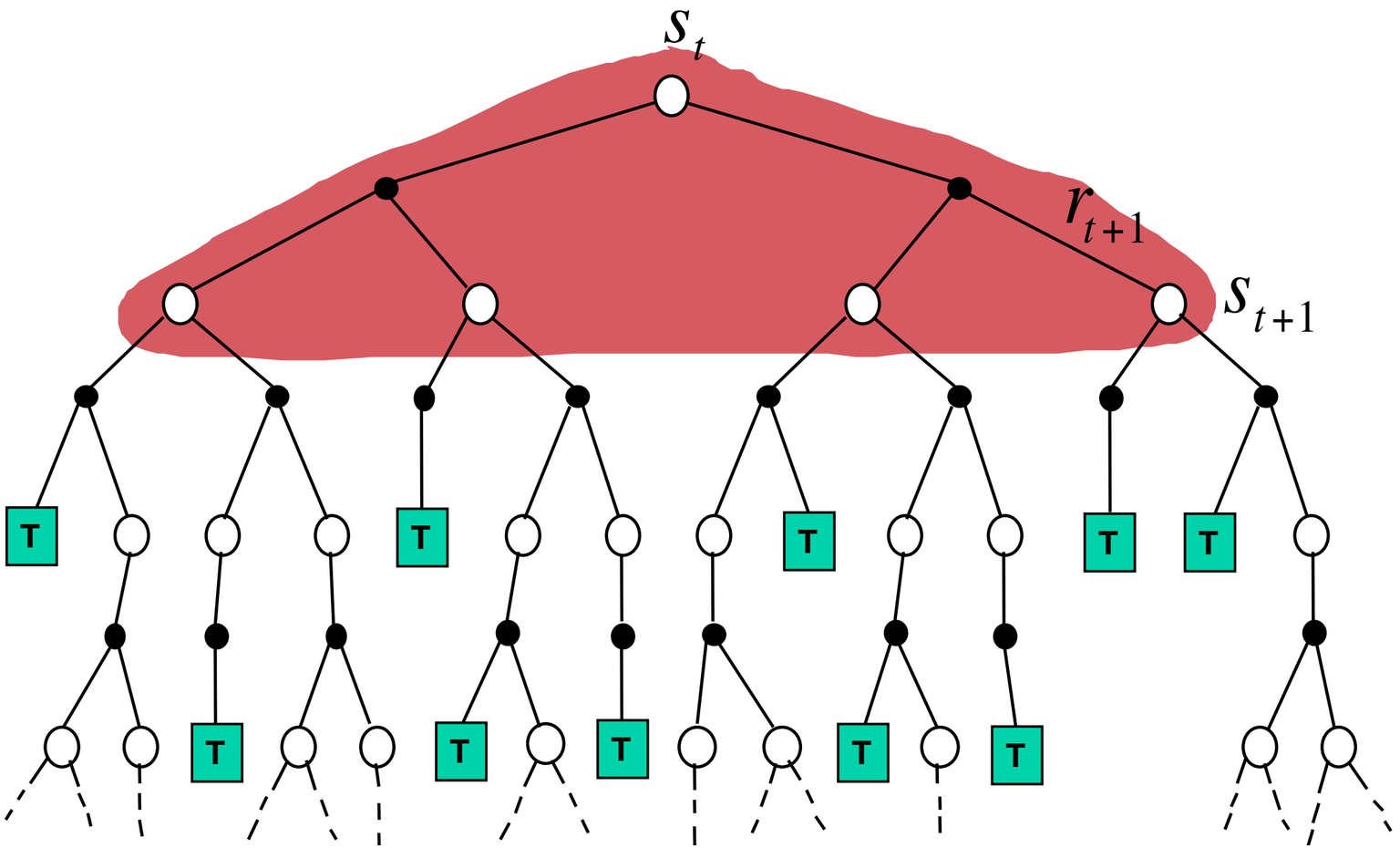
Temporal Difference Backup

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



Dynamic programming backup

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$



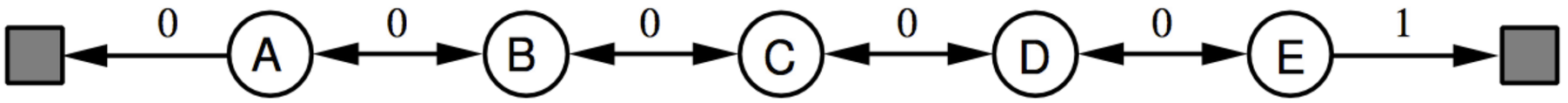
Bias/Variance Trade-off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$ is unbiased estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is biased estimate of $v_\pi(S_t)$

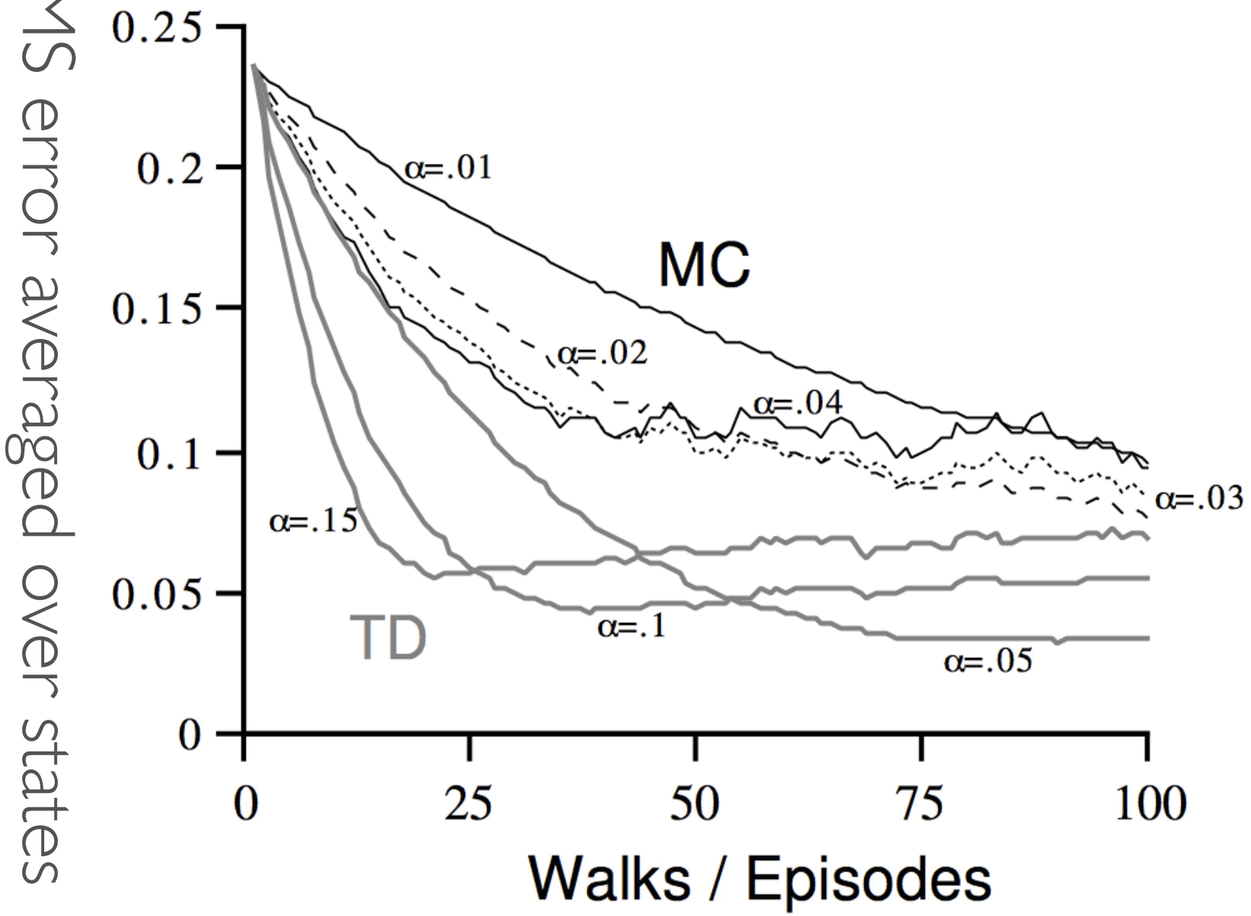
- TD target has much lower variance than the Return G_t
  - G_t depends on *many* random actions, transitions, rewards
  - TD target depends on *one* random action, transition, reward

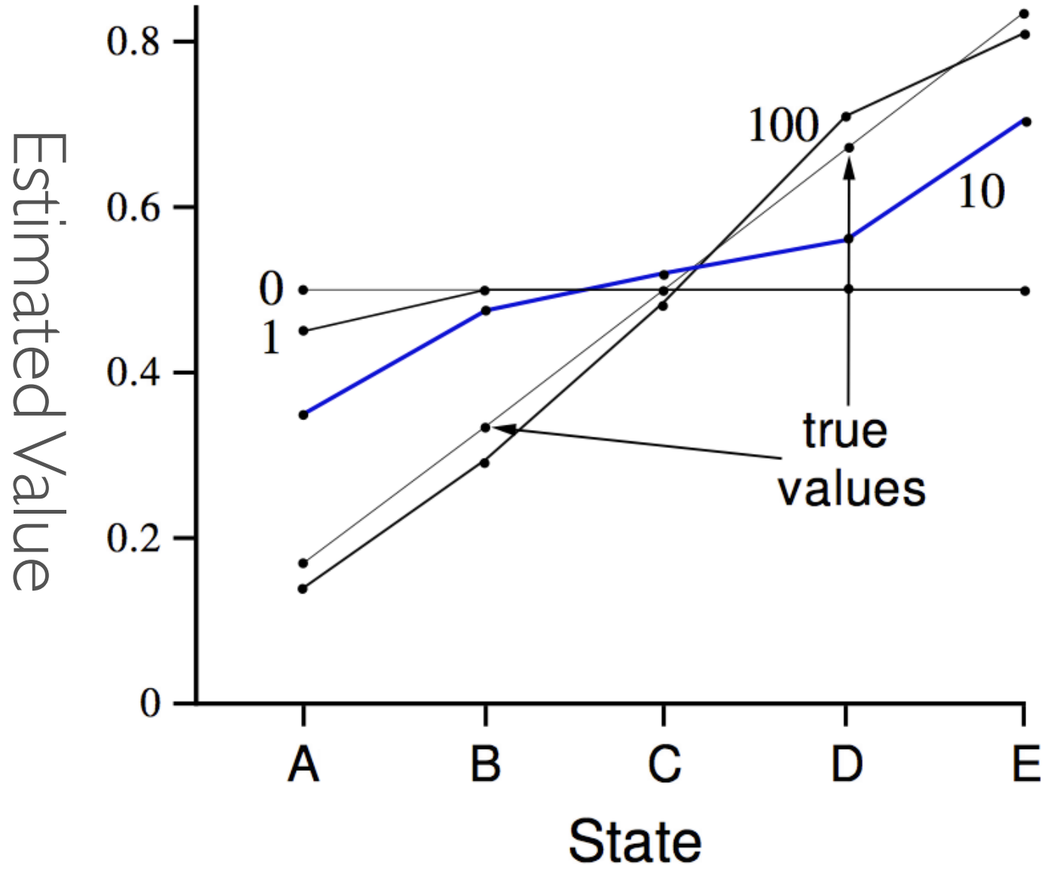
TD vs MC

- MC has high variance, zero bias
  - good convergence properties
  - not sensitive to initial value
  - simpe and intuitive
- TD: low variance, some bias
  - generally much more efficient than MC
  - TD(0) usually converges to $v_\pi(s)$
  - more sensitive to initial value



start



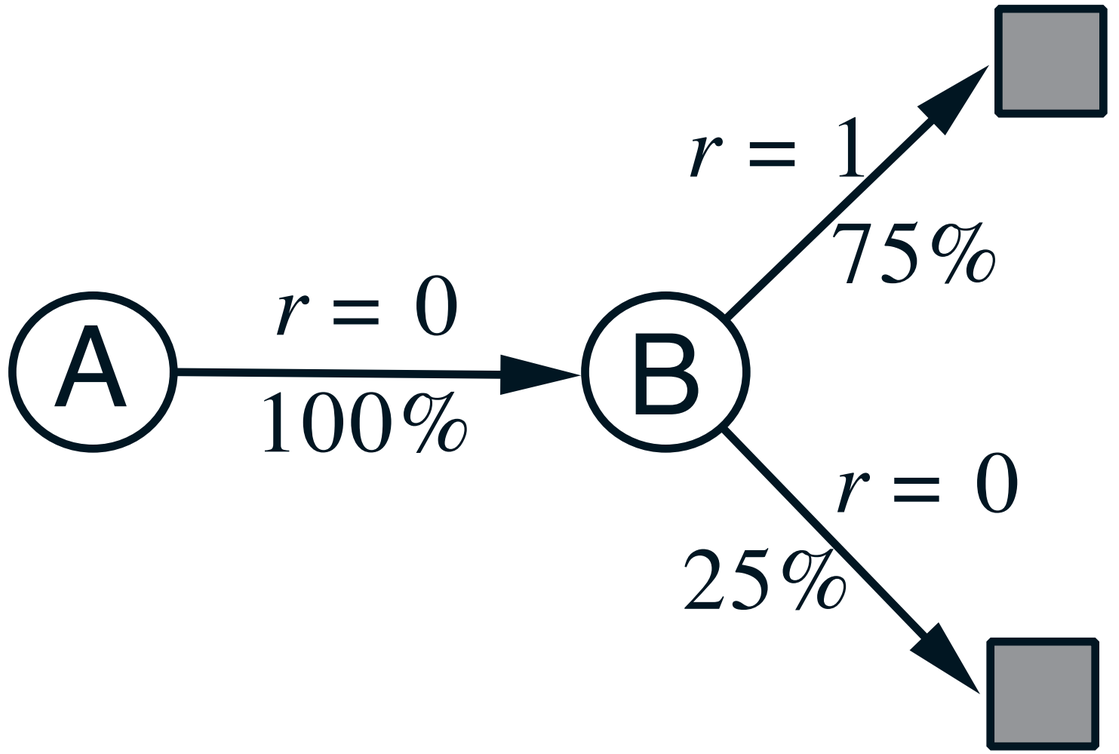


AB Example

- Two states A,B, 8 episodes, no discounting:
  - A,0,B,0
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B,0
- V(B)=?? V(A)=??

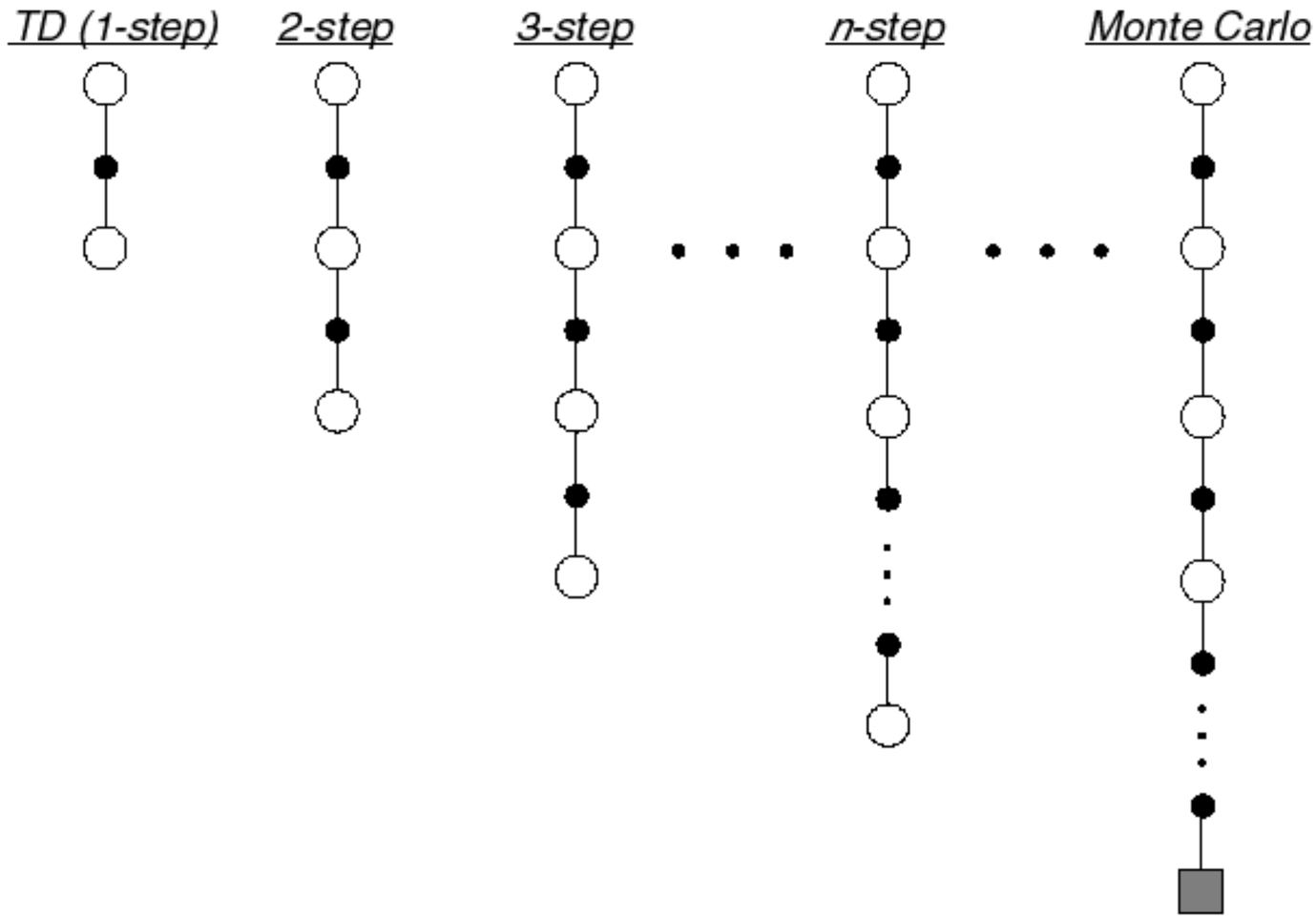
AB Example

- Two states A,B, 8 episodes, no discounting:
  - A,0,B,0
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B,0
- V(B)=?? V(A)=??



n-step predictions

- TD(0) looks one step ahead. What about n steps?



n-step predictions

- TD(0) looks one step ahead. What about n steps? Consider the n-step returns:

$$
\begin{aligned}
n = 1 \qquad & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 \qquad & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots \qquad & \qquad \vdots \\
n = \infty \qquad & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T
\end{aligned}
$$

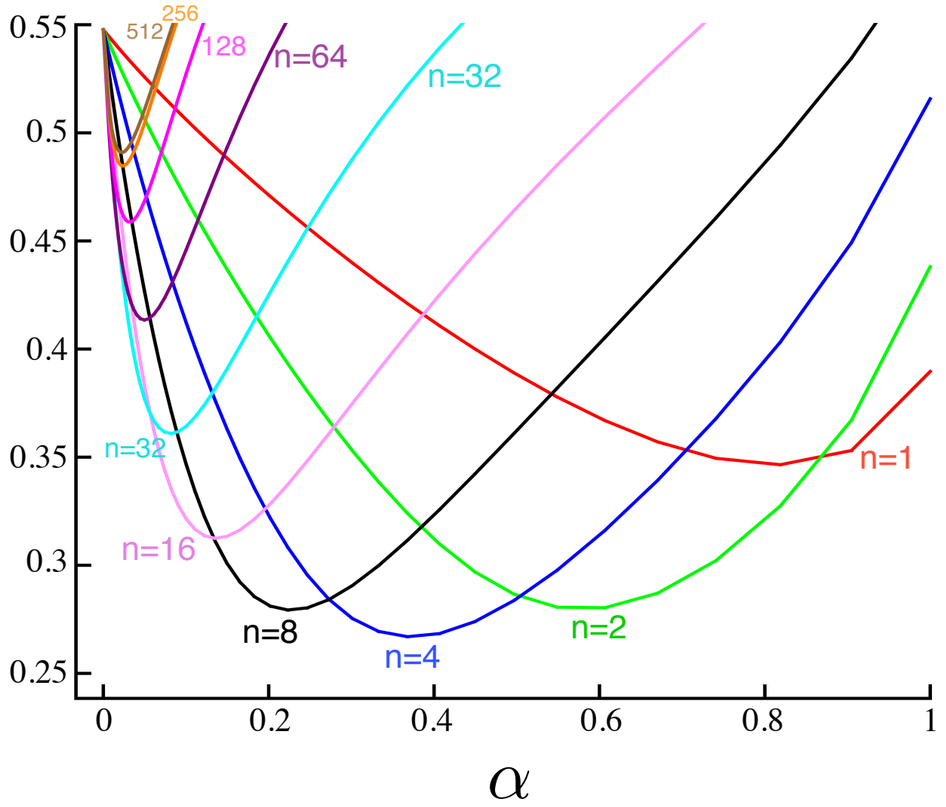
- Define the n-step return $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$
- n-step temporal-difference learning $V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{(n)} - V(S_t) \right)$

finding best n

- random walk as before, 20 states
- testing different step sizes α
- testing different choices of n

- A best n can generally not be identified 😕

Average RMS error over 19 states and first 10 episodes



LUND UNIVERSITY

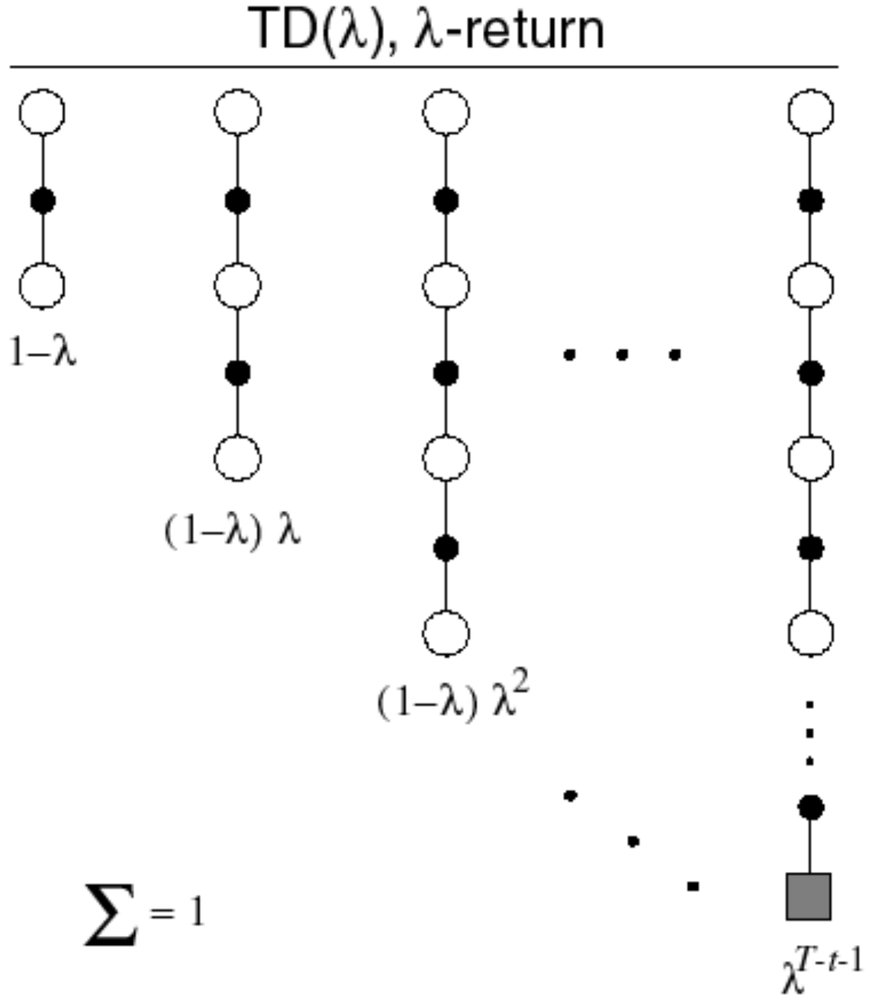
λ return

- The λ-return G_t^λ combines all n-step returns $G_t^{(n)}$
- using weights $(1-\lambda)\lambda^{n-1}$ (geometric series)

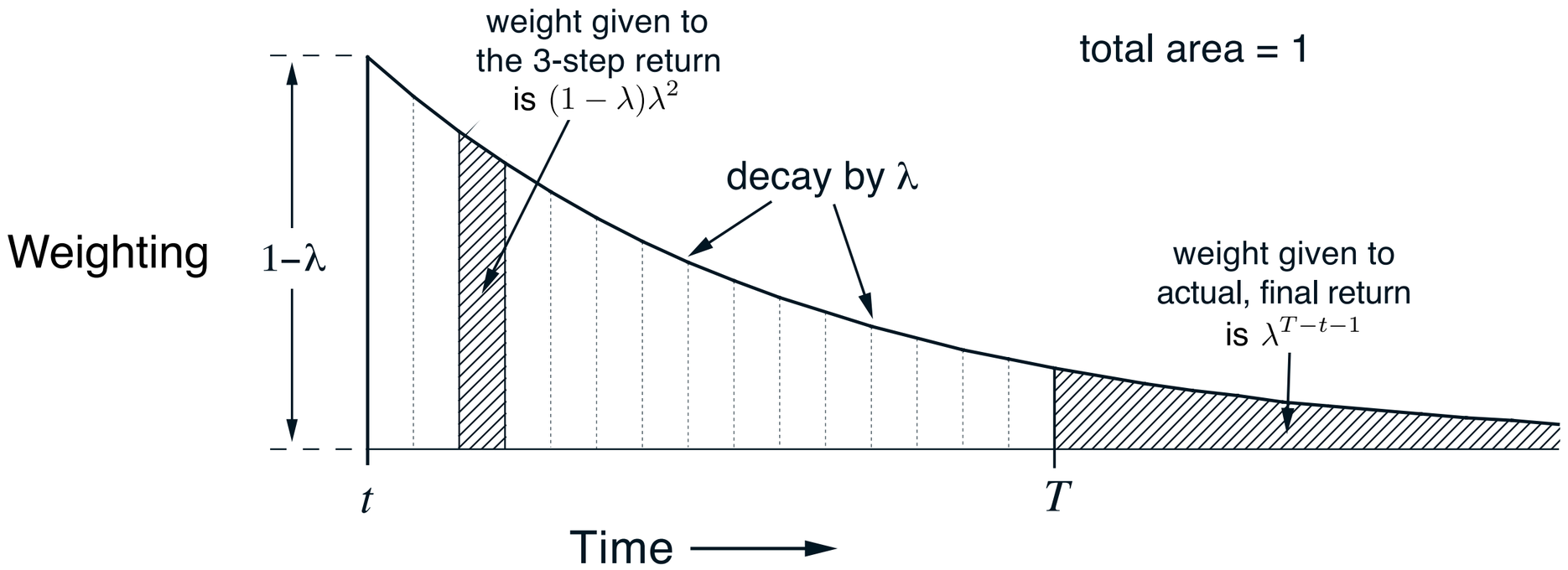
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- Forward-view TD(λ)

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



Return weighting function

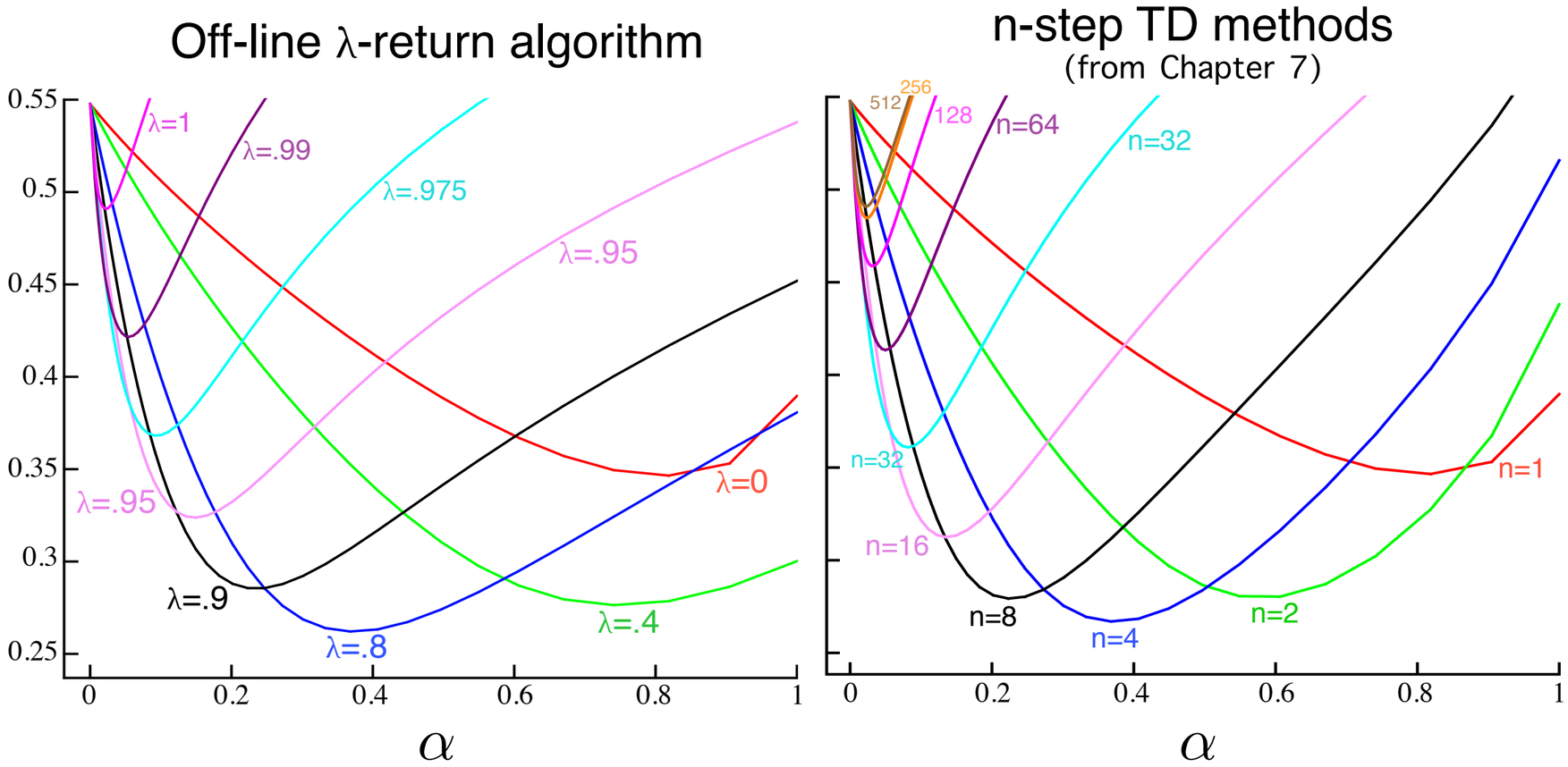


$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

LUND UNIVERSITY

Random Walk α vs. n

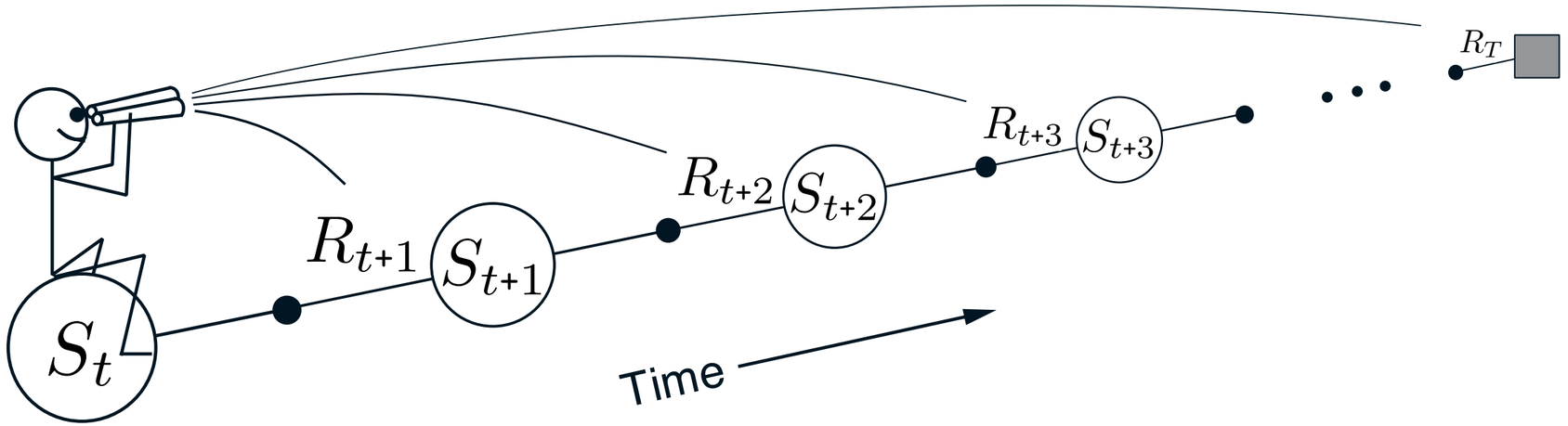
RMS error
at the end
of the episode
over the first
10 episodes



LUND UNIVERSITY

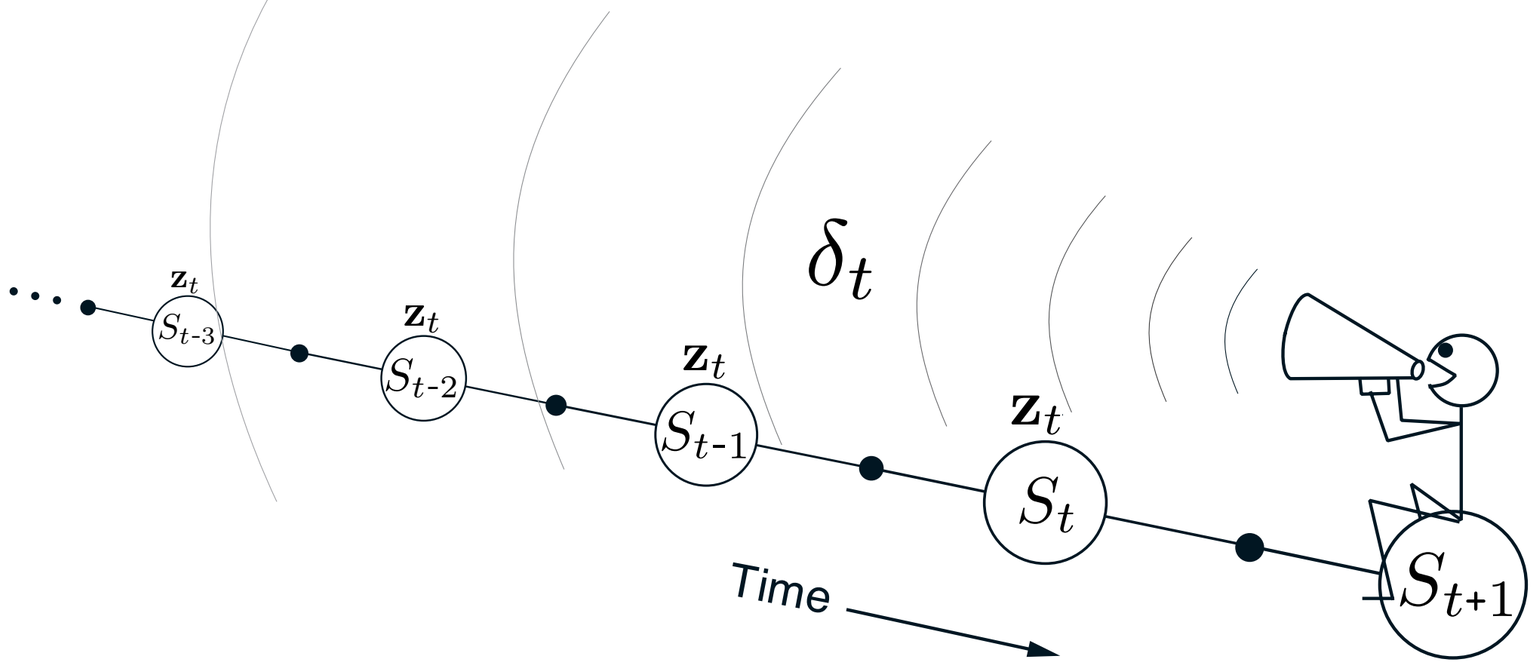
Forward View TD(λ)

- Update the value function towards the λ-return
- Forward-view looks into the future to compute G_t^λ
- Like MC: can only be computed on complete episodes!!
- BUT: the choice of λ is much more robust across applications.



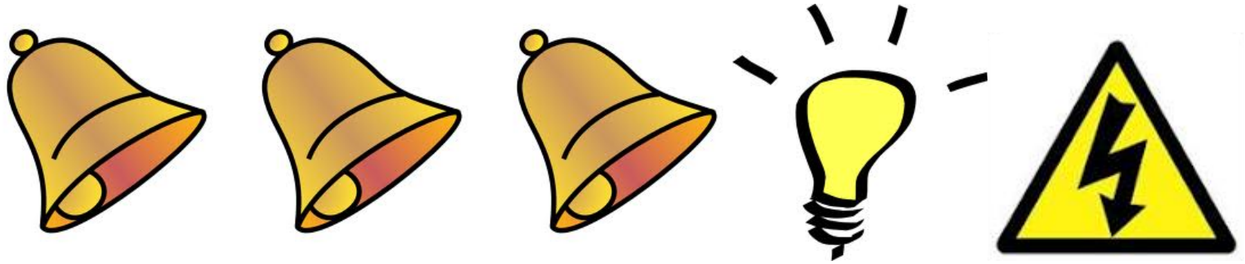
Backward View TD(λ)

- Forward view provides the theory
- Backward view provides the computational efficiency
- Inform backwards how much each state has contributed to the recent reward
- The result: online updates, for every step, from incomplete sequences.



LUND UNIVERSITY

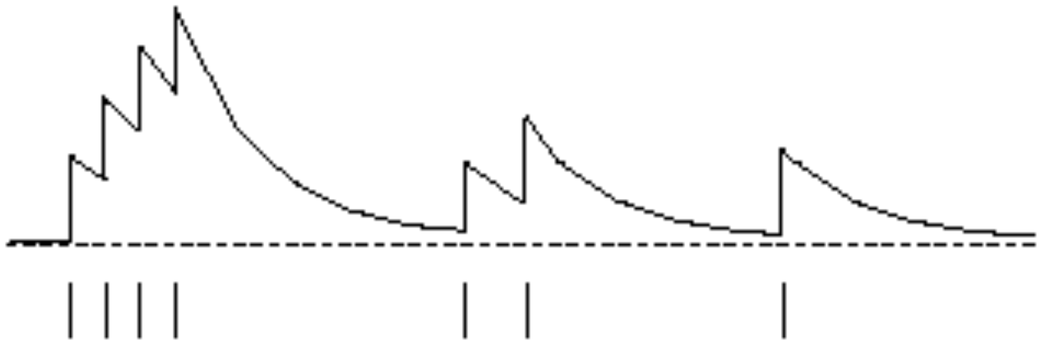
Eligibility Traces



- 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 combine both heuristics

$$E_0(s) = 0$$

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



accumulated eligibility trace

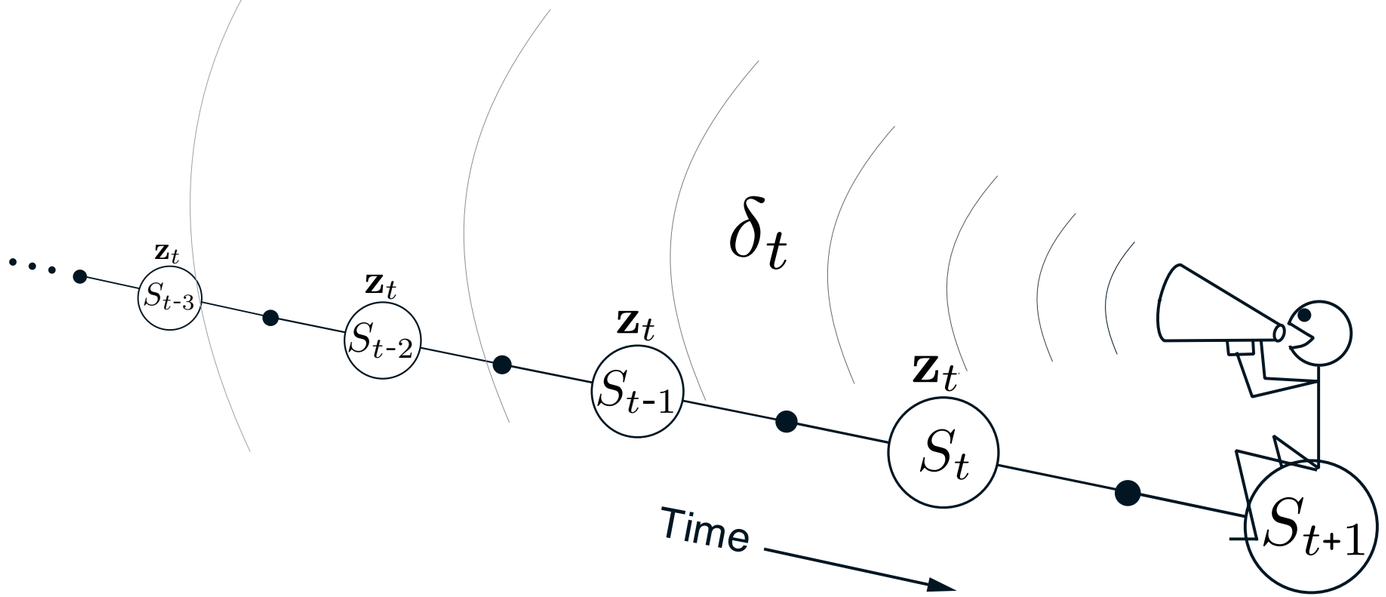
times of visits to a state

LUND UNIVERSITY

Backward View TD(λ)

- Keep an eligibility trace for every state s
- Update value $V(s)$ for every state s
- In proportion to TD-error δ_t and eligibility trace $E_t(s)$

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \qquad V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$



TD(λ), TD(0) and MC

- For $\lambda = 0$ we have exactly TD(0):

- For $\lambda = 1$ we have no decay in the eligibility traces $E_t(s)$. For
  - episodic environments: all episodes are finite
  - offline-updates: the value functions get updated at the end of each episode
  - TD(λ) is exactly MC

TD(λ), TD(0) and MC

- For $\lambda = 0$ we have exactly TD(0):

$$E_t(s) = \lambda E_{t-1} \mathbf{1}(S_t = s) \qquad = \mathbf{1}(S_t = s)$$
$$V(s) \leftarrow V(S) + \alpha \delta_t E_t(s) \qquad = V(S) + \alpha \delta_t$$

- For $\lambda = 1$ we have no decay in the eligibility traces $E_t(s)$. For
  - episodic environments: all episodes are finite
  - offline-updates: the value functions get updated at the end of each episode
  - TD(λ) is exactly MC

TD(λ)

- **Theorem:** *The sum of offline updates is identical for forward-view and backward view TD(λ)*

$$\sum_{t=1}^{T} \alpha \delta_t E_t(s) = \sum_{t=1}^{T} \alpha \left(G_t^\lambda - V(S_t) \right) \mathbf{1}(S_t = s)$$

- Theorem applies for episodic MDPs with offline updates
  - updates are accumulated within each episode
  - updates of the $V(s)$ are applied *at the end* of the episode.
- For online updates the above does not hold!!
  - recent result: a slight change in eligibility traces results in equivalence (Sutton, v. Seijen, ICML2014)

LUND UNIVERSITY

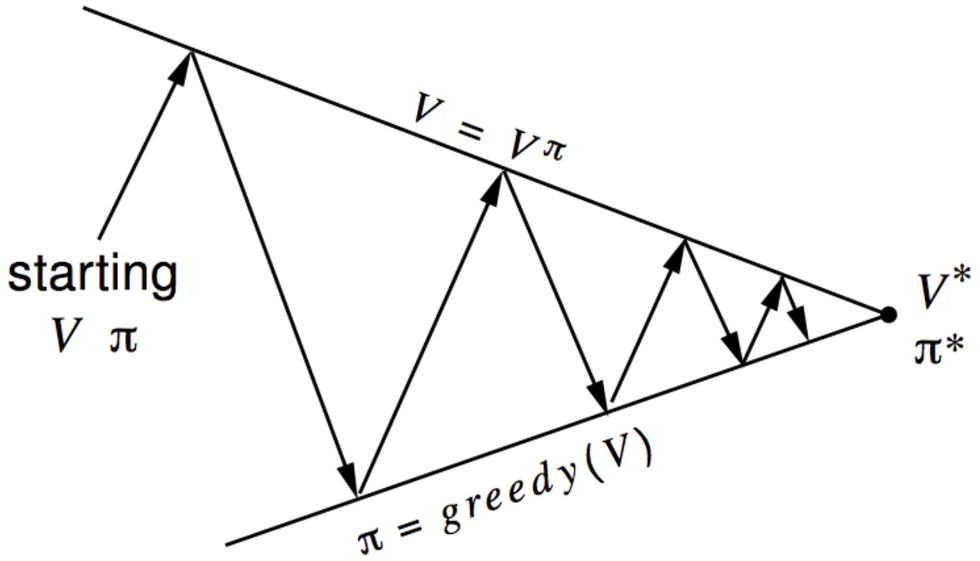
Break

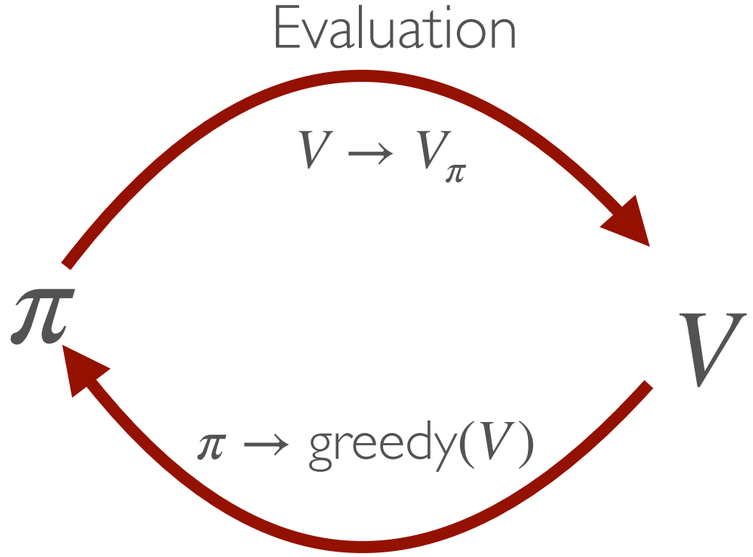
Can we use TD for policy optimisation?

Policy Optimization with TD

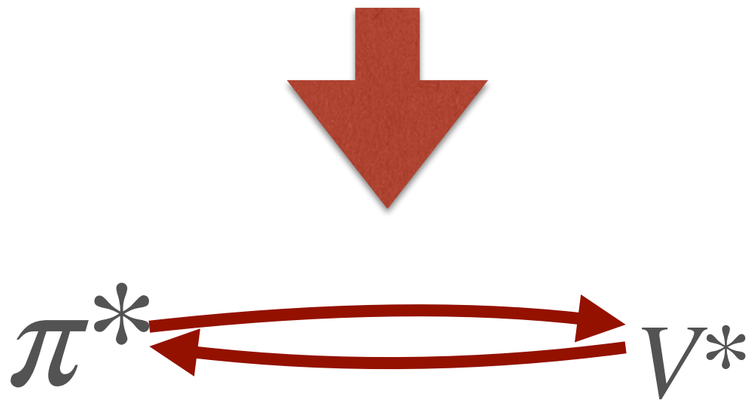
- Similar idea as in the Monte Carlo Learning case
- use TD in the control loop
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy policy
  - Update at every time step
- On-policy learning: Learning by doing: learn π while using π
- Off-policy learning: Observe someone else to learn our own policy μ
  - Best known algorithm in Reinforcement Learning: Q Learning

Generalized Policy Iteration





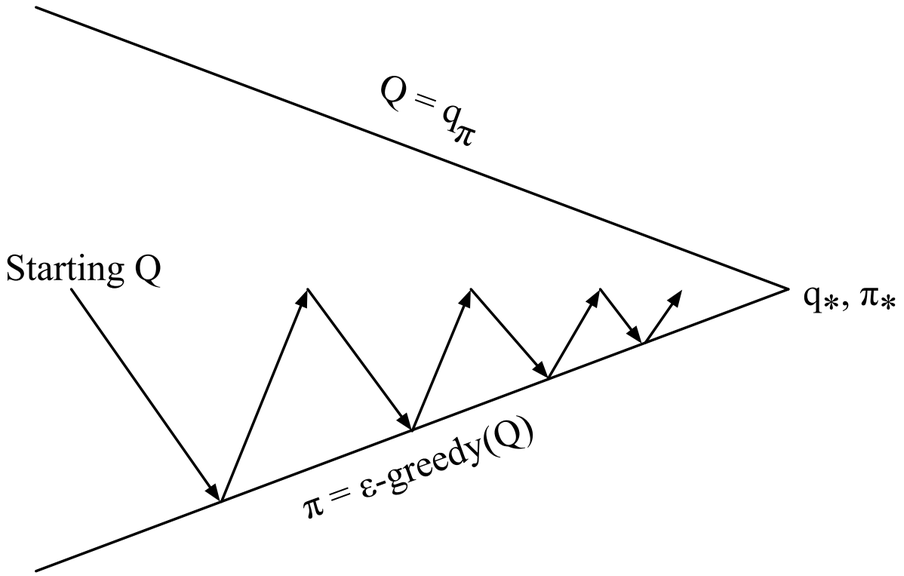
- **Policy Evaluation**: Estimate v_π
  - ***Any*** policy evaluation approach
- **Policy improvement**: Generate $\pi' \geq \pi$
  - ***Any*** policy improvement approach



LUND UNIVERSITY

Monte Carlo Control more efficient

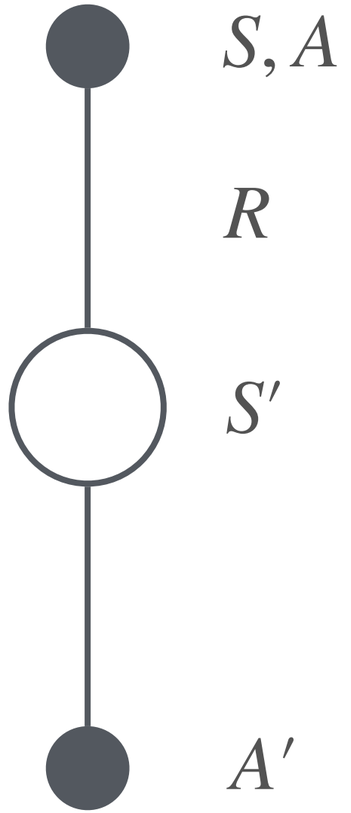
- For every episode:
  - Policy Evaluation:
    - Monte Carlos Policy Evaluation of $Q \approx q_\pi$
  - Policy Improvement:
    - ϵ-Greedy policy improvement



LUND UNIVERSITY

Action-Value Function with SARSA

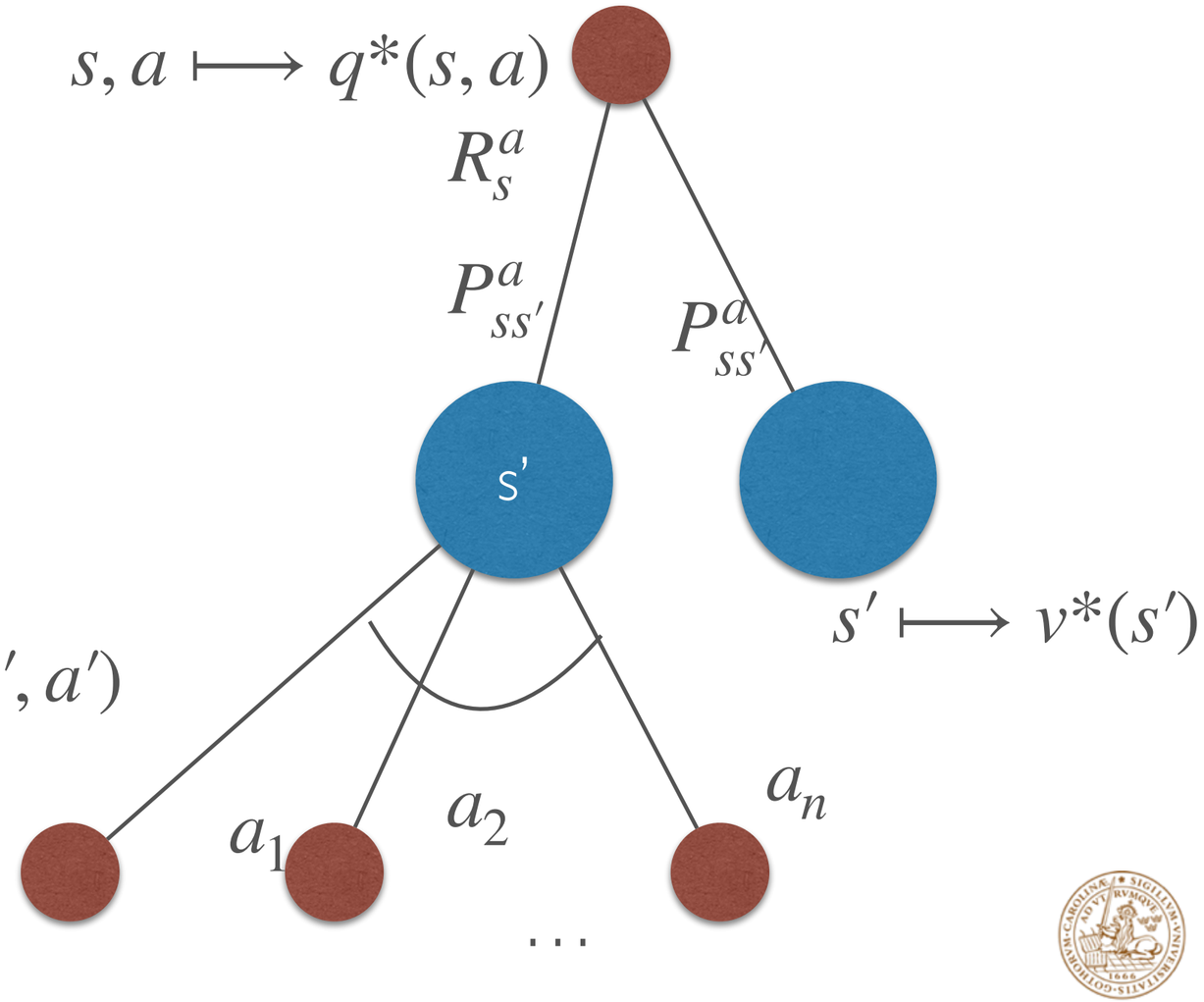
- At **every time step** we improve our policy
- S, A: comes from the episode
- R, S' sampled from the environment
- A' chosen ϵ-greedy: according to our ϵ-greedy policy
- $S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, \ldots$ dynamically generated



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma Q(S', A') - Q(S, A) \right)$$

BELLMAN OPTIMALITY EQUATION

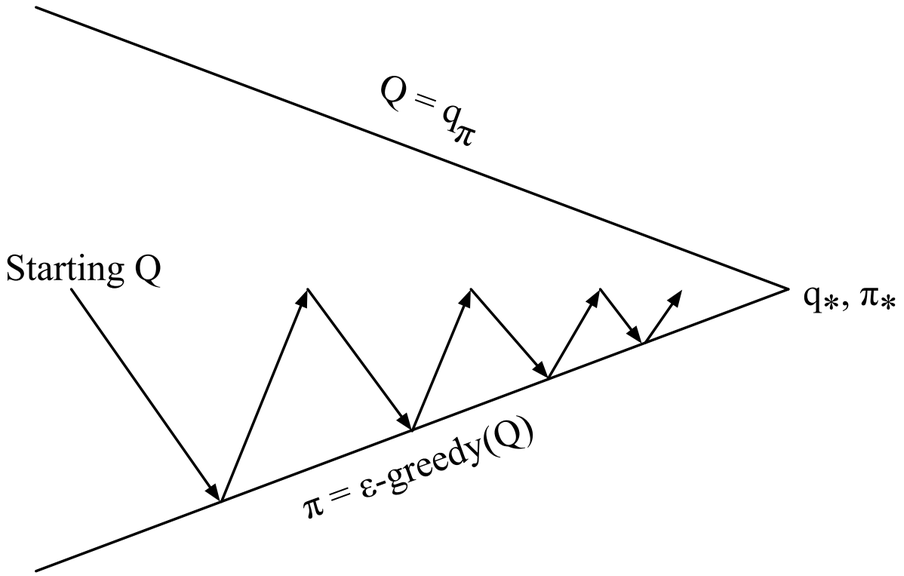
$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a \max_{a' \in \mathbf{A}} q^*(s',a')$$



$$s', a' \longmapsto q^*(s',a')$$

On Policy Control with SARSA

- For **every single time step**:
  - Policy Evaluation:
    - **SARSA** update of $Q \approx q_\pi$
  - Policy Improvement:
    - ϵ-greedy policy improvement
    - no explicit policy necessary



LUND UNIVERSITY

SARSA Algorithm for on-policy control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    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)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

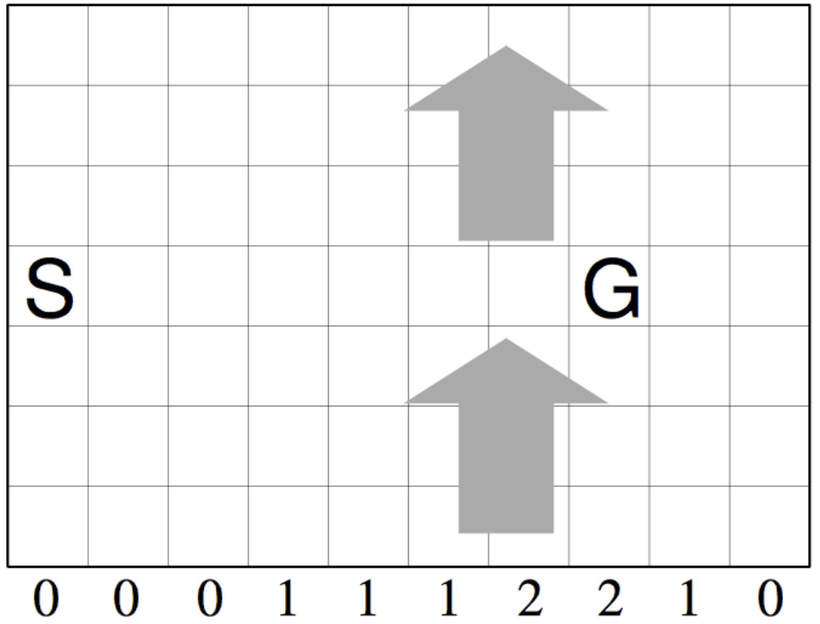
Convergence of SARSA

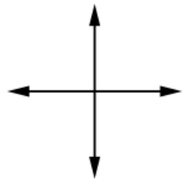
- **Theorem**: *SARSA converges to the optimal action-value function* $Q(S, A) \rightarrow q^*(s, a)$, *under the following conditions:*
  - *GLIE sequence of policies* $\pi_t(a, s)$
  - *Robbins-Monro sequence of step-sizes* α_t

$$\sum_{t=1}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

  - $\alpha_k = \frac{1}{k}$ *works*
  - SARSA almost always works, no matter GLIE and α_t 😀

Windy Gridworld example



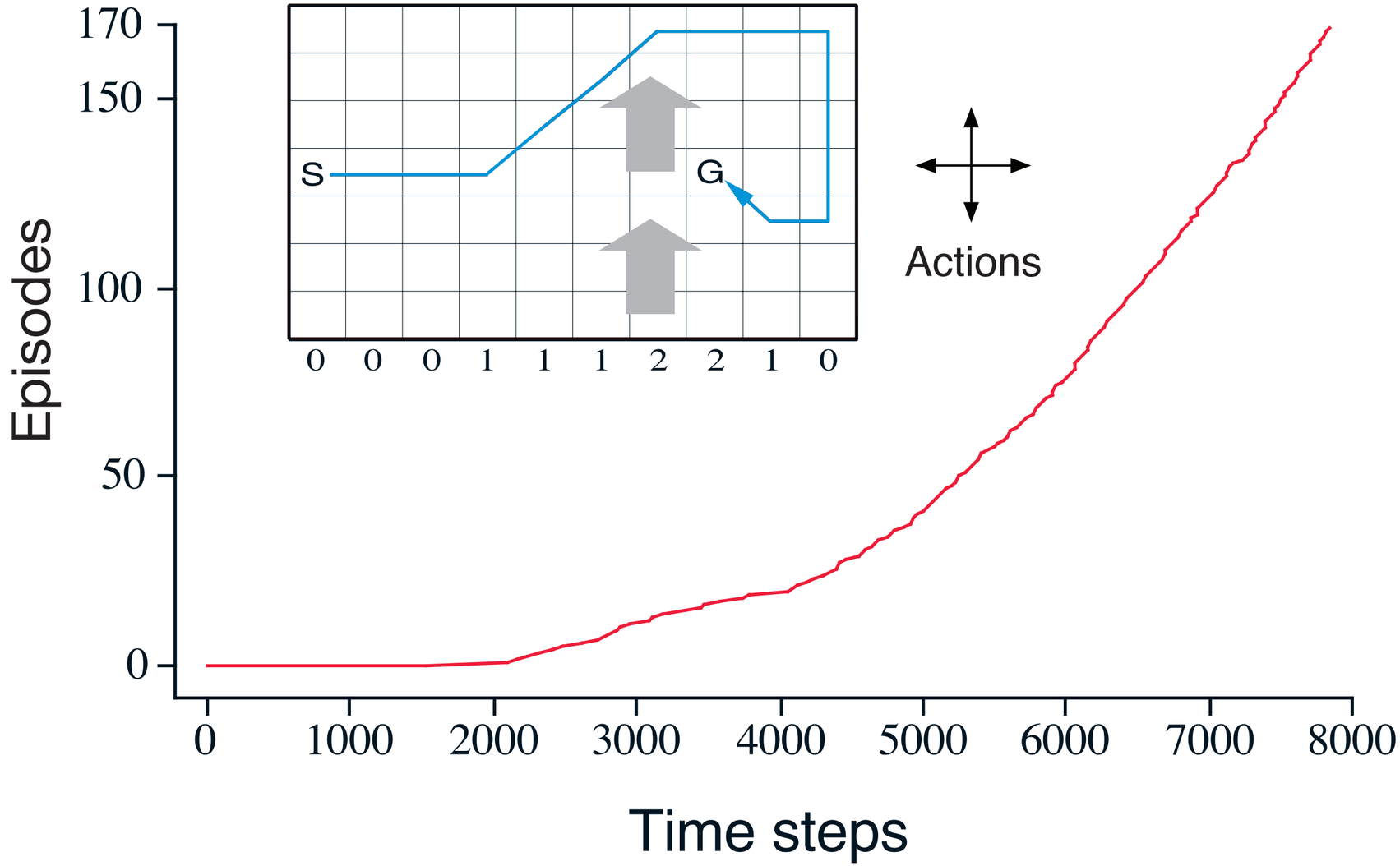


standard moves

wind pushing upward n gridpoints

- Reward $= -1$ per time step until goal is reached
- undiscounted: $\gamma = 1$

SARSA on the Windy Gridworld



LUND UNIVERSITY

n-step SARSA

- SARSA looks one step ahead. What about n steps? Consider the n-step returns:

$$
\begin{aligned}
n = 1 \qquad & q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \\
n = 2 \qquad & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2,A_{t+2}}) \\
\vdots \qquad & \qquad \vdots \\
n = \infty \qquad & q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T
\end{aligned}
$$

- Define the n-step Q-return $q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$
- n-step SARSA update of $Q(s, a)$ toward the n-step Q-return

$$
Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right)
$$

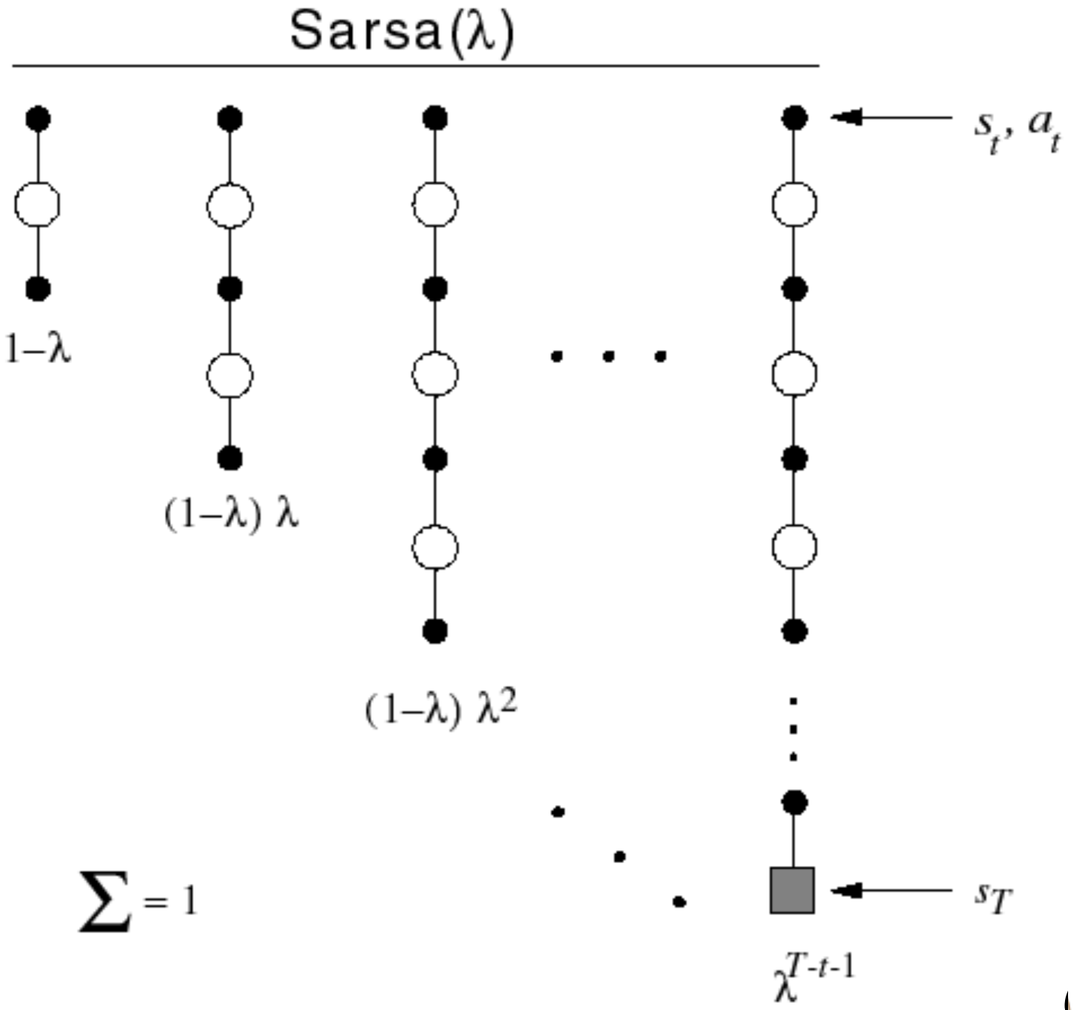
Forward View

- The q^λ-return combines all n-step returns Q-returns q_t^n
- using weights $(1-\lambda)\lambda^{n-1}$

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

- Forward-view SARSA(λ)

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left(q_t^\lambda - Q(S_t,A_t)\right)$$



Backward View SARSA(λ)

- Like TD(λ) use **eligibility traces** in an online algorithm
- SARSA(λ) has one eligibility trace for each state-action pair

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

- $Q(s,a)$ is updated for every state s and action a.
- In proportion to TD-error δ_t and eligibility trace $E_t(s,a)$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$$

Sarsa

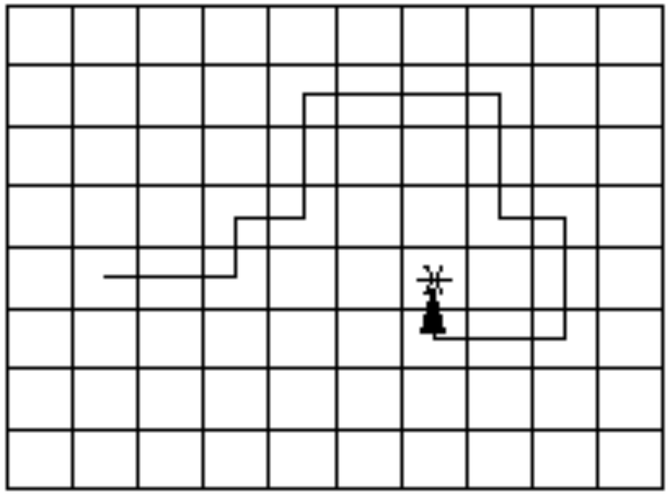
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy 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 $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$
      $E(s, a) \leftarrow \gamma \lambda E(s, a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

Sarsa

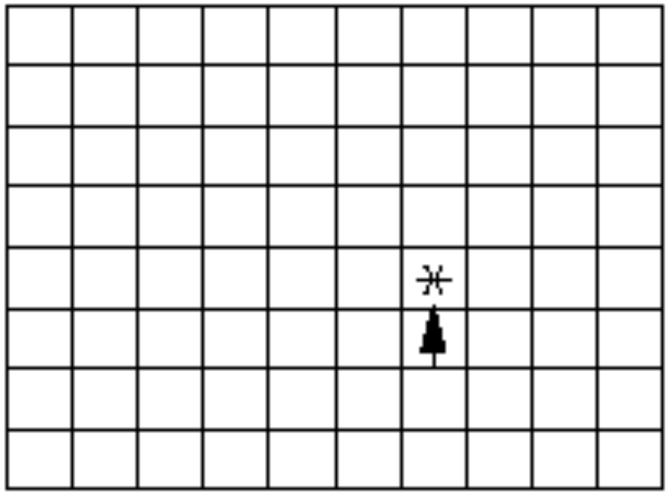
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy 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 $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$
      $E(s, a) \leftarrow \gamma \lambda E(s, a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

SARSA(λ) Gridworld Example

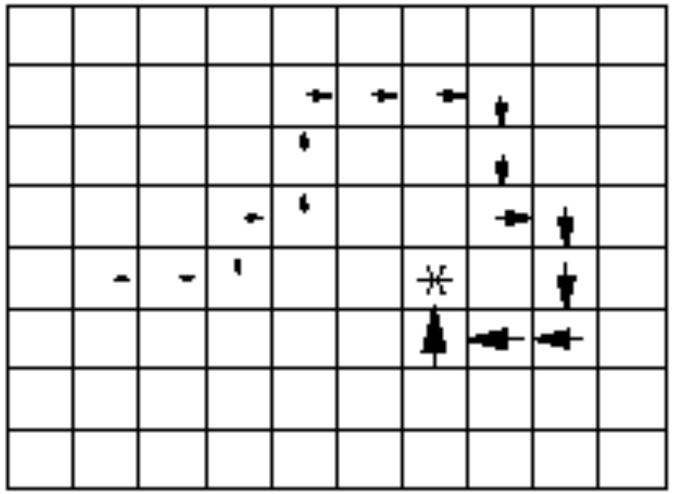
Path taken



Action value increased by one step SARSA



Action values increased by SARSA(λ) with $\lambda = 0.9$



LUND UNIVERSITY

Off-policy learning

- Evaluate target policy $\pi(a, s)$ to compute $v_\pi(s)$ or $Q_\pi(s, a)$ while following behaviour policy $\mu(a, s)$

$$(S_1, A_1, R_2, S_2, A_2, R_3, \ldots) \sim \mu$$

- Why is this important?
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies $\pi_1, \pi_2, \pi_3, \ldots, \pi_{t-1}$
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy

Q Learning

- off-policy learning not possible with MC
- off-policy learning of action-values $Q(s, a)$
- next action in the demonstration is chosen using behaviour policy: $A_{t+1} \sim \mu(A, S_t)$
  - but we consider **alternative** successor action: $A' \sim \pi(A, S_t)$
  - and update $Q(S_t, A_t)$ towards value of the **alternative** action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

LUND UNIVERSITY

off-policy Control with Q-Learning

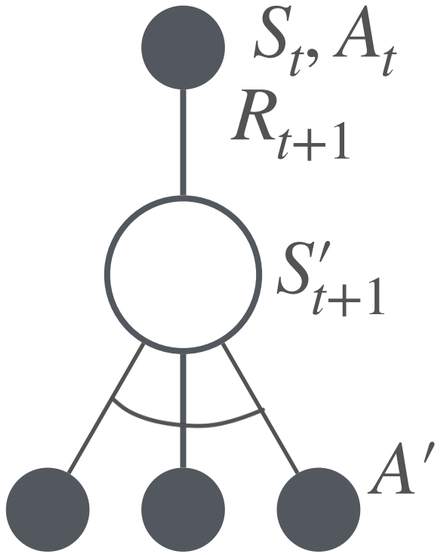
- We now allow both behaviour and target policies to **improve**
- The behaviour policy μ is, e.g., **ϵ-greedy** w.r.t. $Q(S, A)$
- S_{t+1} is found using an action A_{t+1}according to policy μ
- The target policy π is **greedy** w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a') = A'$$

- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ = & R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a')) \\ = & R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

Q-Learning Control Algorithm



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right)$$

- Theorem: Q-learning control converges to the optimal action-value function $Q(s, a) \rightarrow q^*(s, a)$

Q-Learning Algorithm for off-policy control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$

Repeat (for each episode):

  Initialize S

  Repeat (for each step of episode):

    Choose A from S using policy derived from Q (e.g., ε-greedy)

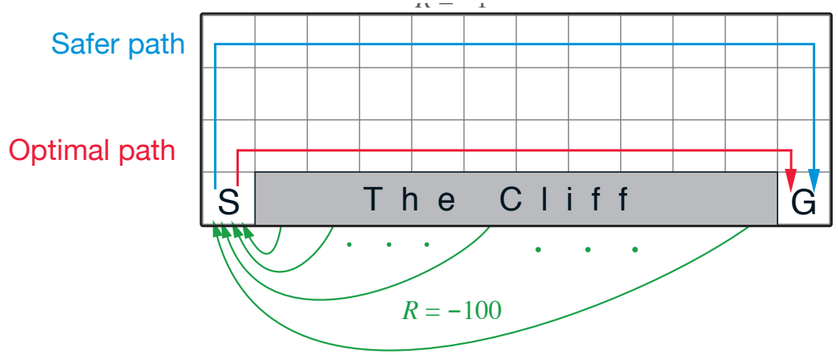
    Take action A, observe R, S'

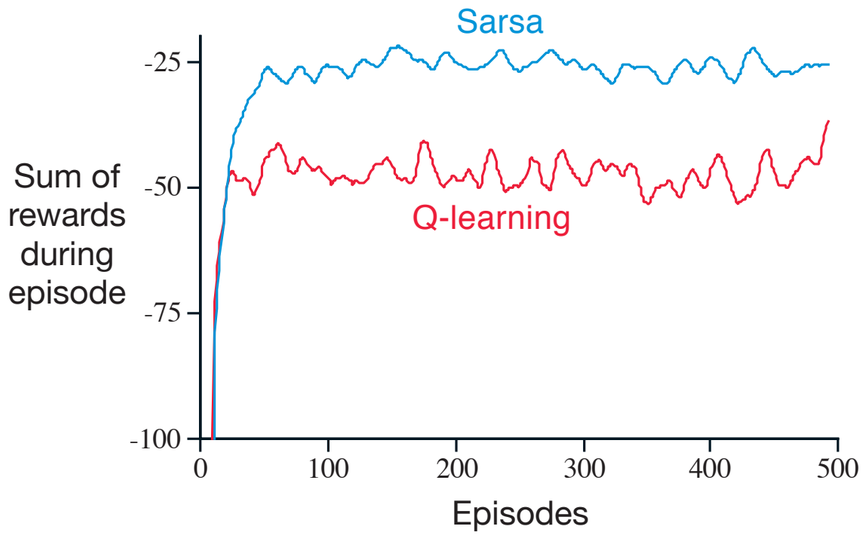
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

    $S \leftarrow S'$;

  until S is terminal

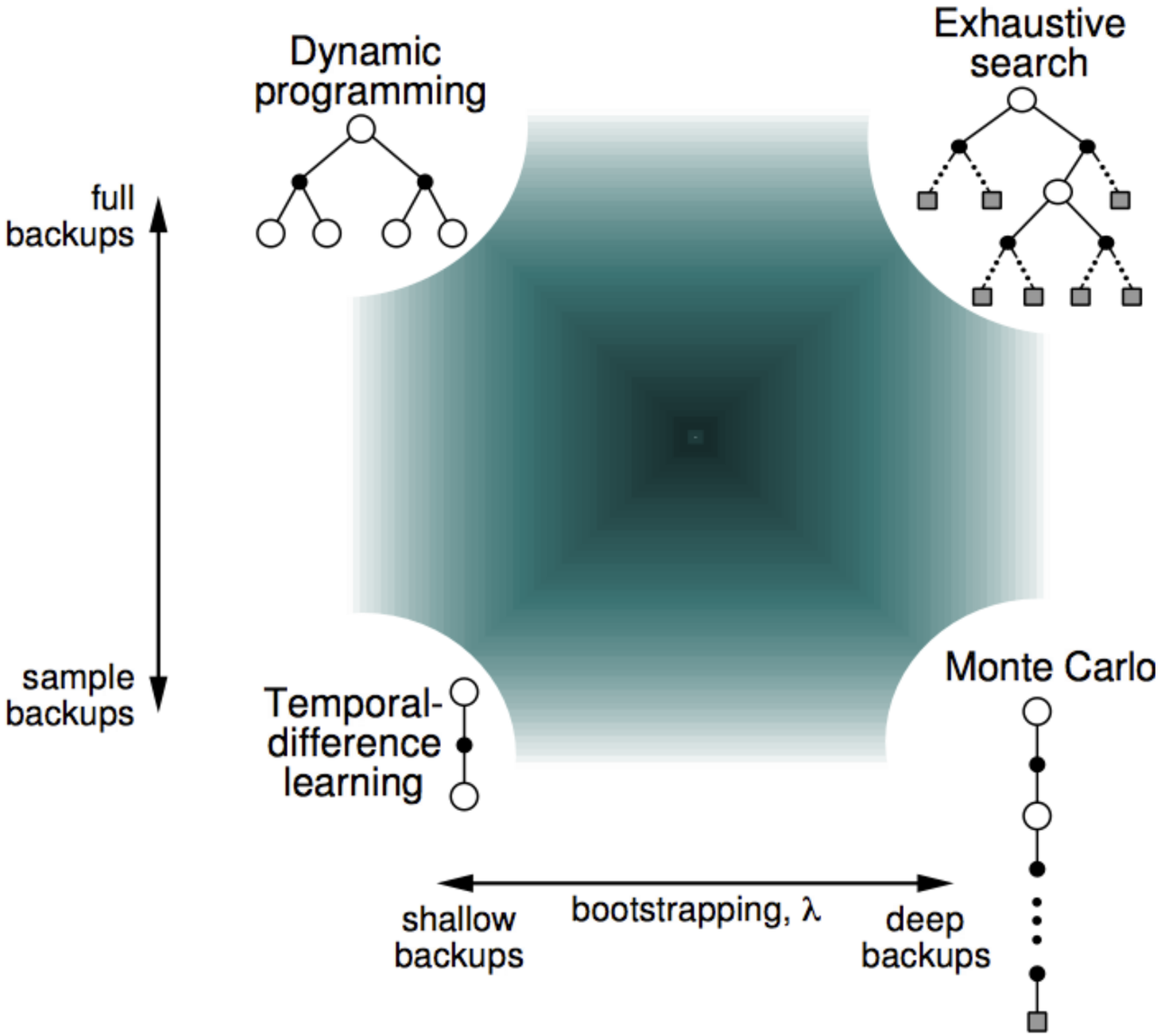
Cliff Walking Example





LUND UNIVERSITY

Unified View



Summary and Conclusions

- MDP,
- DP, Policy Iteration, Value Iteration; MC-Learning
- TD, TD(λ), SARSA, Q-Learning
  - n-step return, n-step q-return, eligibility traces

Deep Reinforcement Learning

- Function approximation of Q, V, π
- Monte Carlo Tree Search
- Pairing Deep Learning with Reinforcement learning.
- Open master thesis possibilities to explore Deep RL

Comments about Machine Learning