Temporal Difference Learning

KENNETH TRAN

Principal Research Engineer, MSR AI

Temporal Difference Learning

Policy Evaluation

Outline

- Intro to model-free learning
- Monte Carlo Learning
- Temporal Difference Learning
- TD(λ)

Revisit notations

- Episode

$$S_0, A_0, R_1, S_1, A_1, R_2, \dots$$

state

action

reward

- Return

Discount rate, e.g. 0.9

$$\begin{aligned} G_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

- state-value function

$$\begin{aligned} v_\pi(s) &\stackrel{\text{def}}{=} \mathbf{E}_\pi[G_t | S_t = s] \\ &= \mathbf{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbf{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \,| S_t = s] \end{aligned}$$

True value of state s
under policy π

Recap: Dynamic Programming

- Bellman equation

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right)$$

- DP: solve a known MDP

Policy Evaluation

Monte Carlo Learning

Overview

- MC methods learn from episodes of experience
- MC is model-free: no knowledge of MDP equations
- MC learns from complete episodes
- MC uses the simplest possible idea: use empirical mean to approximate the expected value
- Caveat: can only apply MC to episodic MDPs

  (i.e. all episodes must terminate)

MC for Policy Evaluation

- Goal: learn v_π from episodes of experience under π

$$S_1, A_1, R_2, \ldots, S_k$$

- Recall: return is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

- Recall: value function is the expected return

$$v_\pi(s) = \mathbf{E}[G_t | S_t = s]$$

- MC policy evaluation uses *empirical mean* return instead of *expected* return

MC for Policy Evaluation

To evaluate $v_\pi(s)$

- Sample episodes of experience under π
- Every time t that state s is visited in an episode
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers: $\mathrm{V}(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow 0$

Incremental MC Updates

- Update $V(s)$ incrementally after episode

$$S_1, A_1, R_2, \dots, S_T$$

- For each state s, with return G_t
  - $N(S_t) \leftarrow N(S_t) + 1$
  - $\mathrm{V}(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$

Policy Evaluation

Temporal Difference Learning

Temporal Difference Learning

- Like MC, TD is model-free: no knowledge of MDP transition/rewards
- Unlike MC, TD learns from incomplete episodes, by *bootstrapping*
- TD updates a guess towards a guess

MC and TD

- **Goal: policy evaluation**

  For a given policy v_π, compute the state-value function v_π

- Recall: simple every-visit Monte Carlo method

$$V(S_t) \leftarrow V(S_t) + \underbrace{\alpha}_{\text{step-size parameter}}[\underbrace{G_t}_{\textbf{target:}\text{ the actual return after time } t} - V(S_t)]$$

- Simplest temporal-difference learning method: TD(0)

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\textbf{target:}\text{ an estimate of the return}} - V(S_t)]$$

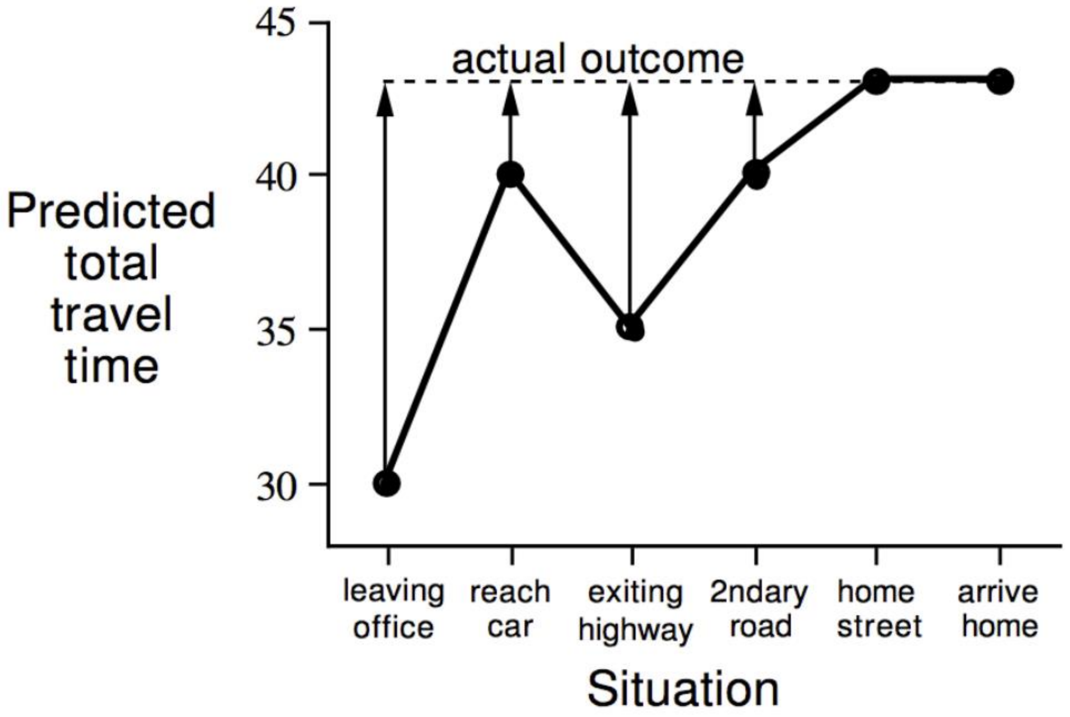
  $\delta_t \stackrel{\text{def}}{=} R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Example: Driving Home

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 example: MC vs TD

Changes recommended by Monte Carlo methods (α=1)



Changes recommended by TD methods (α=1)

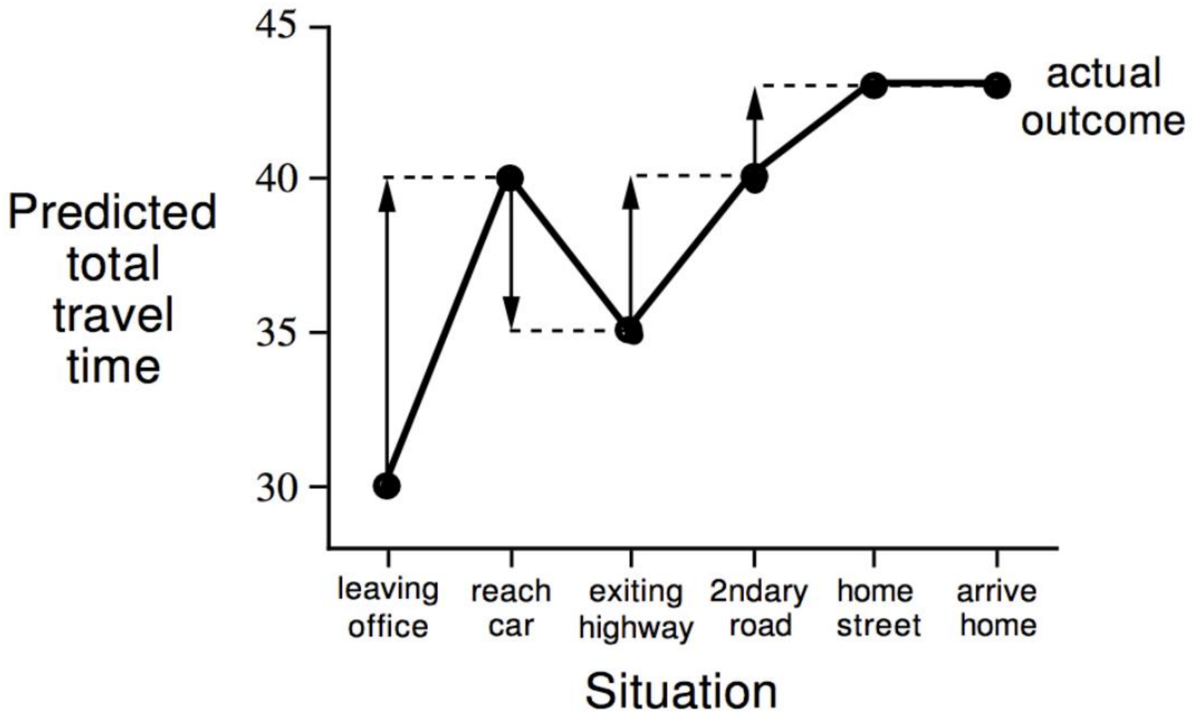


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Pros and Cons of MC vs. TD (1)

- TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Bias/Variance Trade-Off

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

Pros and Cons of MC vs. TD (2)

- MC has high variance, zero bias
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - More sensitive to initial value

Random Walk Example

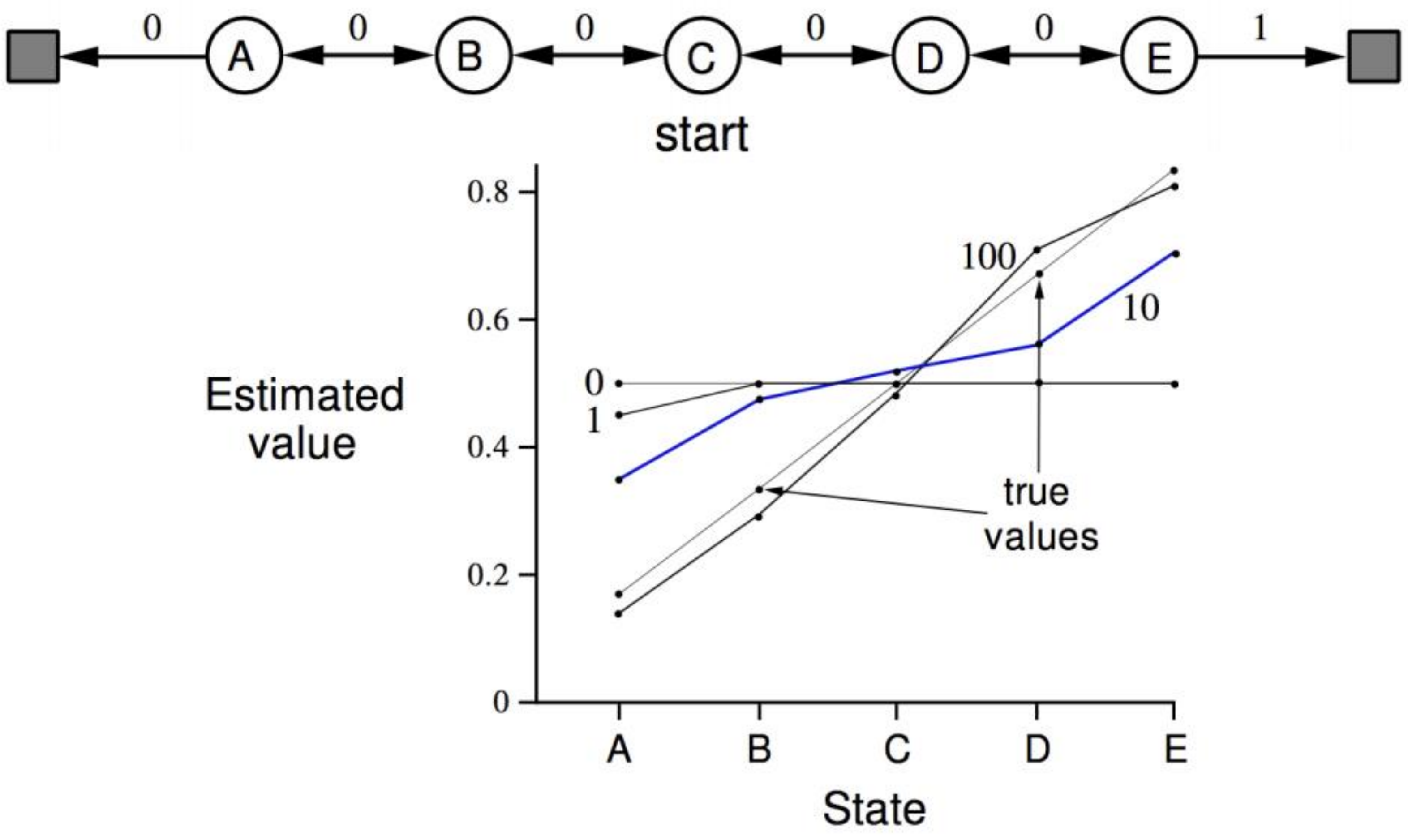


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Random Walk: MC vs. TD

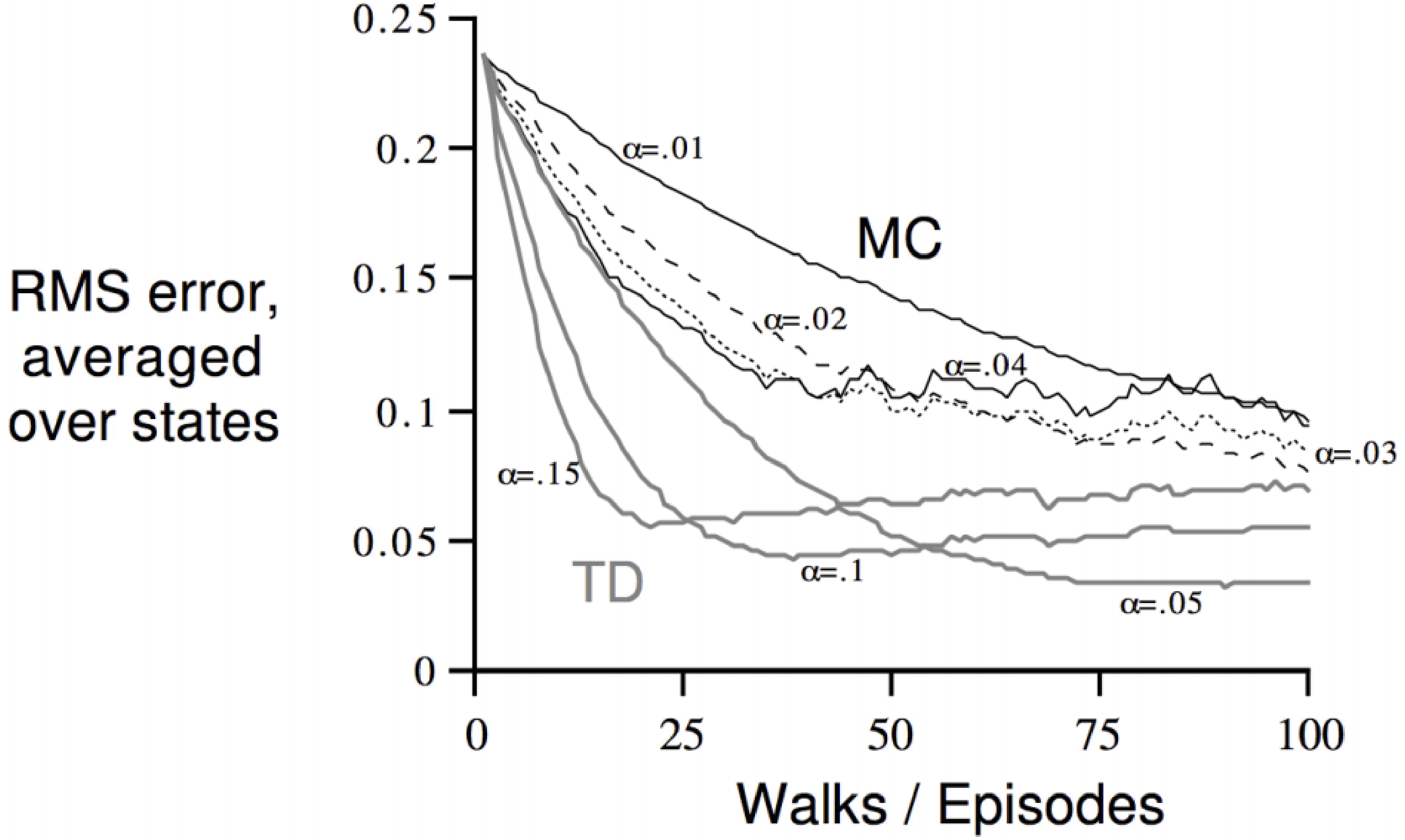


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Batch MC and TD

- MC and TD converge: $V\ (s)\ \rightarrow\ v_\pi(s)$ as experience $\rightarrow \infty$
- But what about batch solution for finite experience?

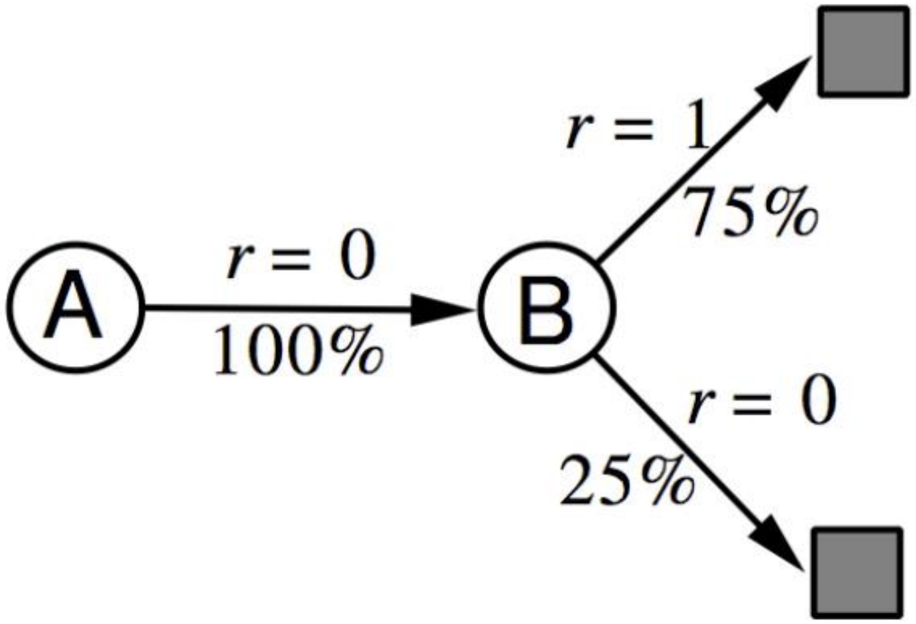
$$s_1^1, a_1^1, r_2^1, \dots, s_{T_1}^1$$
$$\dots$$
$$s_1^K, a_1^K, r_2^K, \dots, s_{T_1}^K$$

  - e.g. Repeatedly sample episode $k\ \in\ [1, K]$
  - Apply MC or TD(0) to episode k
- Will they converge to the same value function?

AB Example

Two states *A*, *B*; no discounting; 8 episodes of experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



What is $V(A), V(B)$?

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Certainty Equivalence

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

$$\sum_{k=1}^{K}\sum_{t=1}^{T_k}\left(G_t^k - V\left(s_t^k\right)\right)^2$$

  - In the AB example, $V(A) = 0$
- TD(0) converges to solution of max likelihood Markov model
  - Solution to the MDP $\langle S, A, P, R, \gamma \rangle$ that best fits the data
  - In the AB example, $V(A) = 0.75$

Pros and Cons of MC vs. TD (3)

- TD exploits Markov property → more efficient
- MC does not exploit Markov property

Unified View of

- Exhaustive Search
- Dynamic Programming
- Monte Carlo
- and Temporal Difference Learning

Monte-Carlo Backup

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

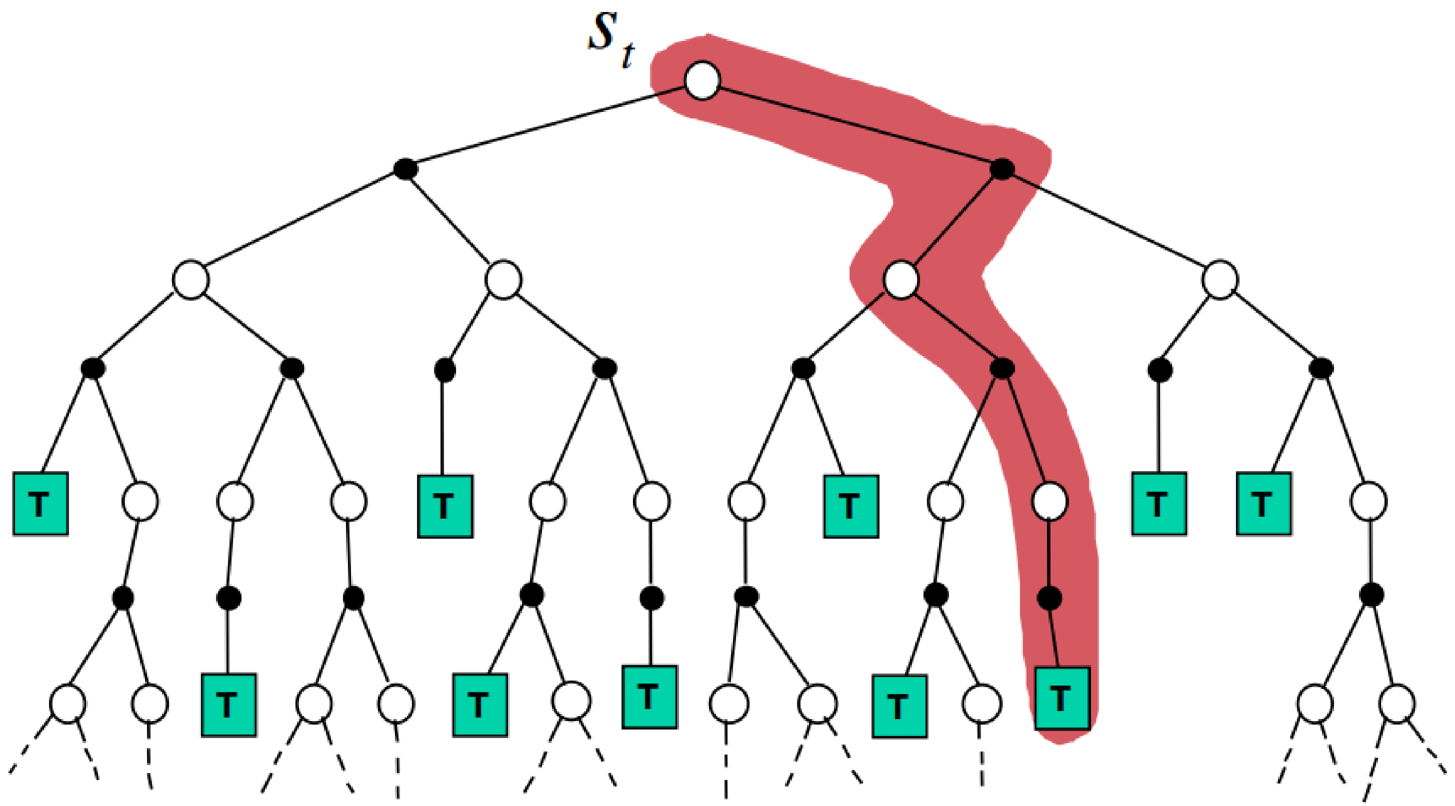


Image Credit: David Silver, Model-Free Prediction, UCL Course on RL

Temporal-Difference Backup

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

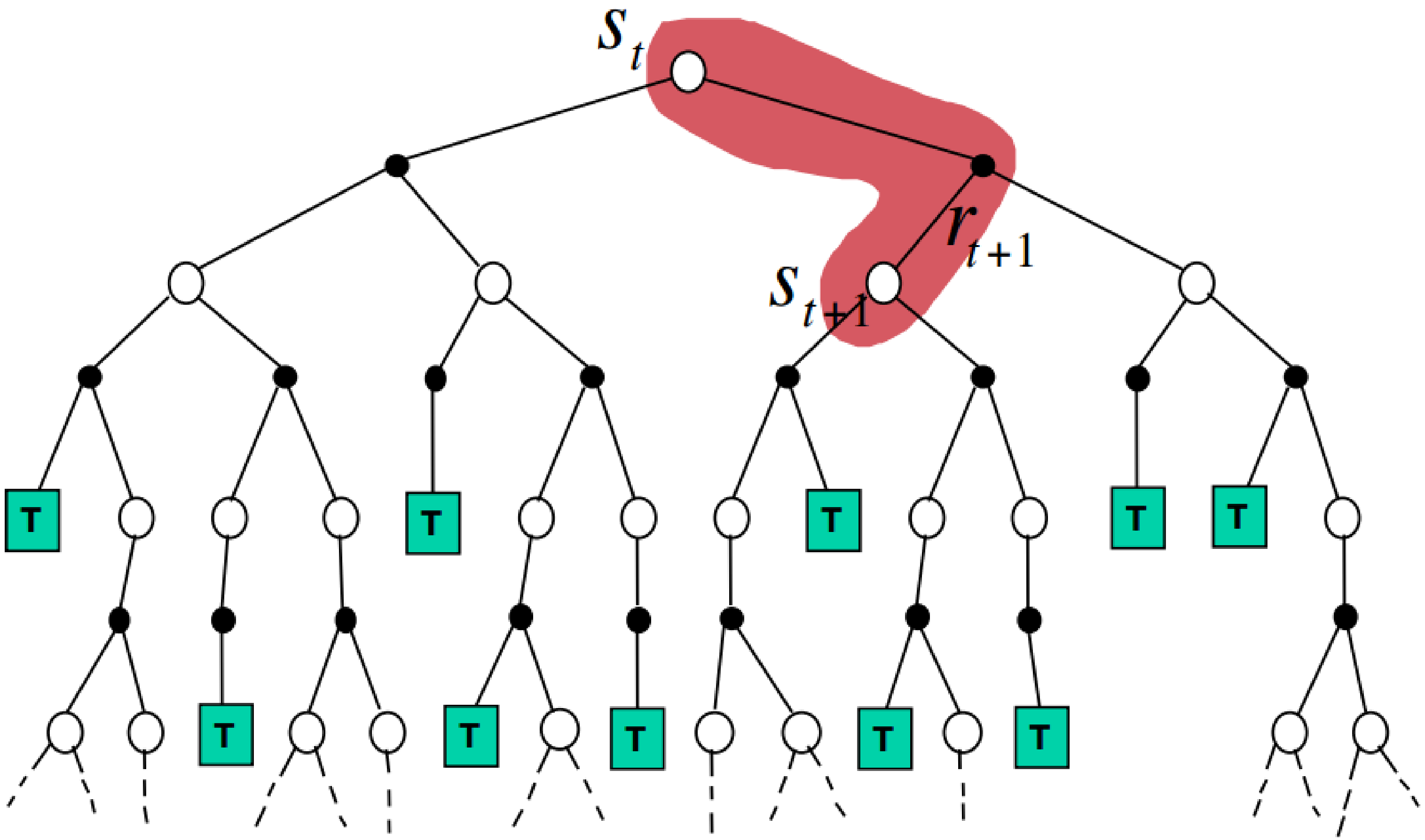


Image Credit: David Silver, Model-Free Prediction, UCL Course on RL

Dynamic Programming Backup

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

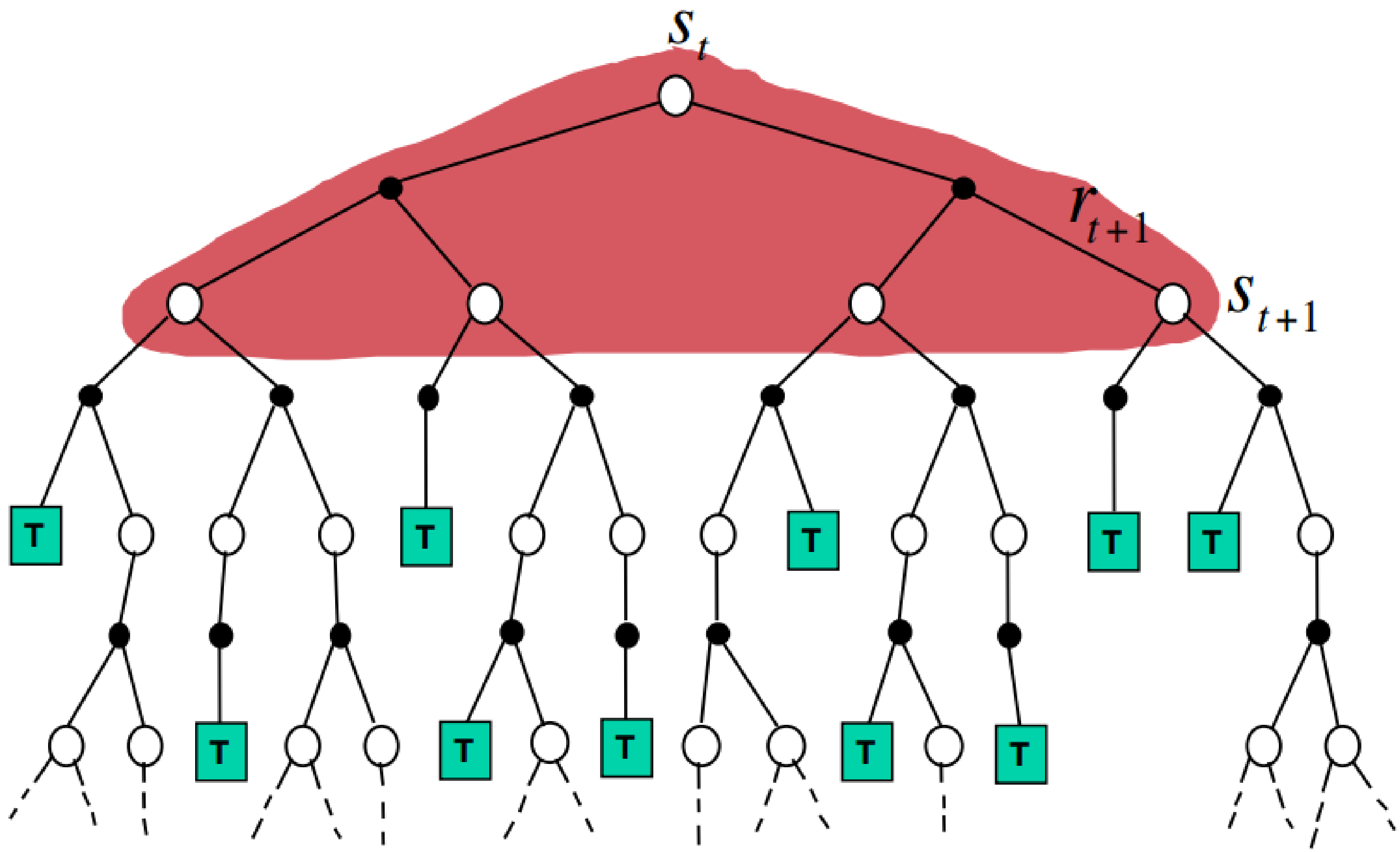


Image Credit: David Silver, Model-Free Prediction, UCL Course on RL

Bootstrapping and Sampling

- **Bootstrapping**: update involves an *estimate*
  - MC does not bootstrap: it learns from complete episodes ❌

    MC must wait until end of episode before return is known
  - DP bootstrap ✓
  - TD bootstrap ✓
- **Sampling**: update does not require computing exact *expectation*
  - MC samples ✓
  - DP does not sample ❌
  - TD samples ✓

Unified View of Reinforcement Learning

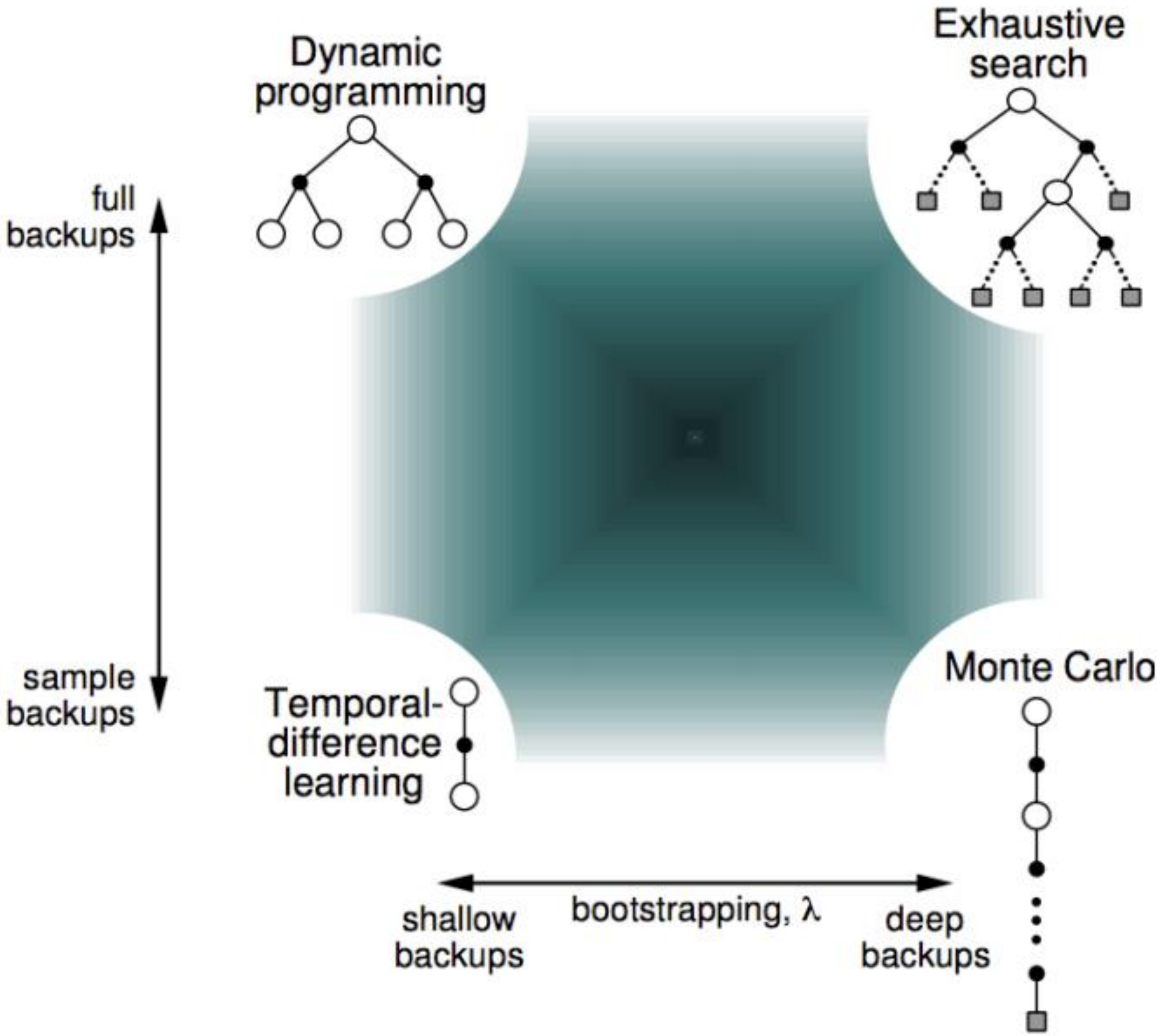


Image Credit: David Silver, Model-Free Prediction, UCL Course on RL

Policy Evaluation

TD(λ)

n-Step Prediction

Let TD target look *n* steps into the future

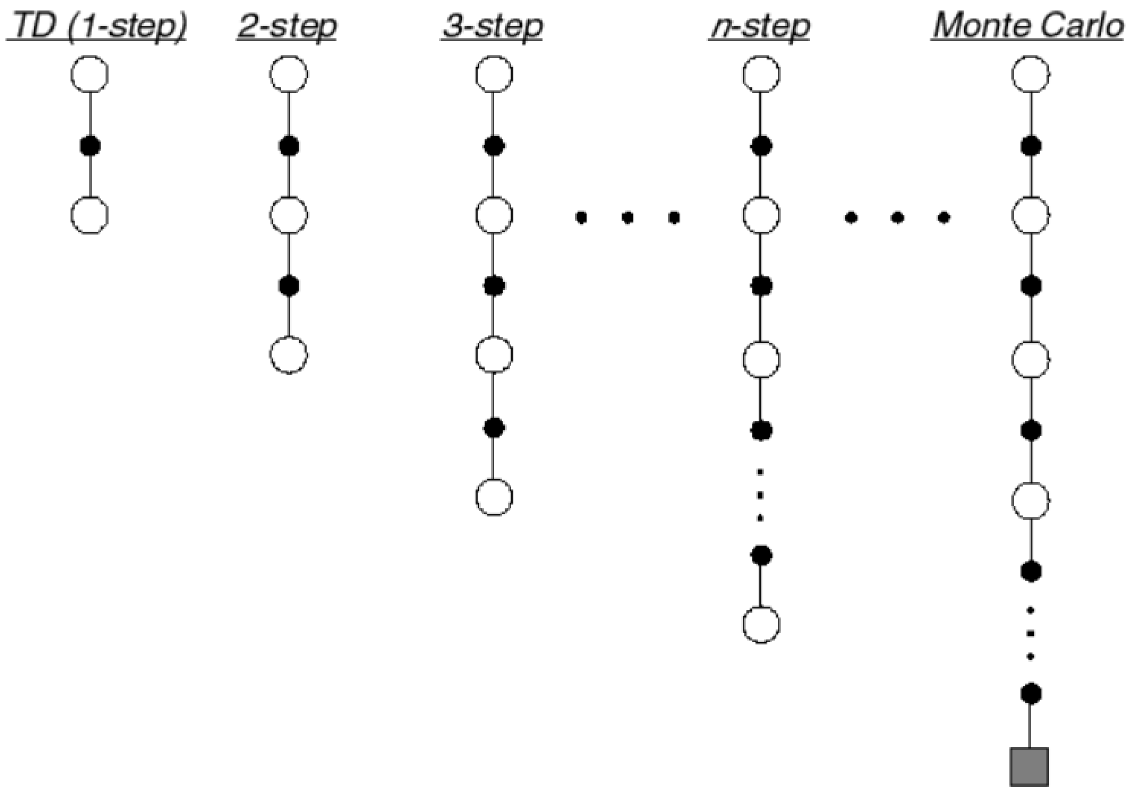


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

n-Step Return

- Consider the following n-step returns for $n = 1, 2, \infty$:

$$n = 1 \quad (TD) \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$

$$n = 2 \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma V(S_{t+2})$$

$$\dots$$

$$n = \infty \quad (MC) \quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Define the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n-step temporal-difference learning

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

Large Random Walk Example

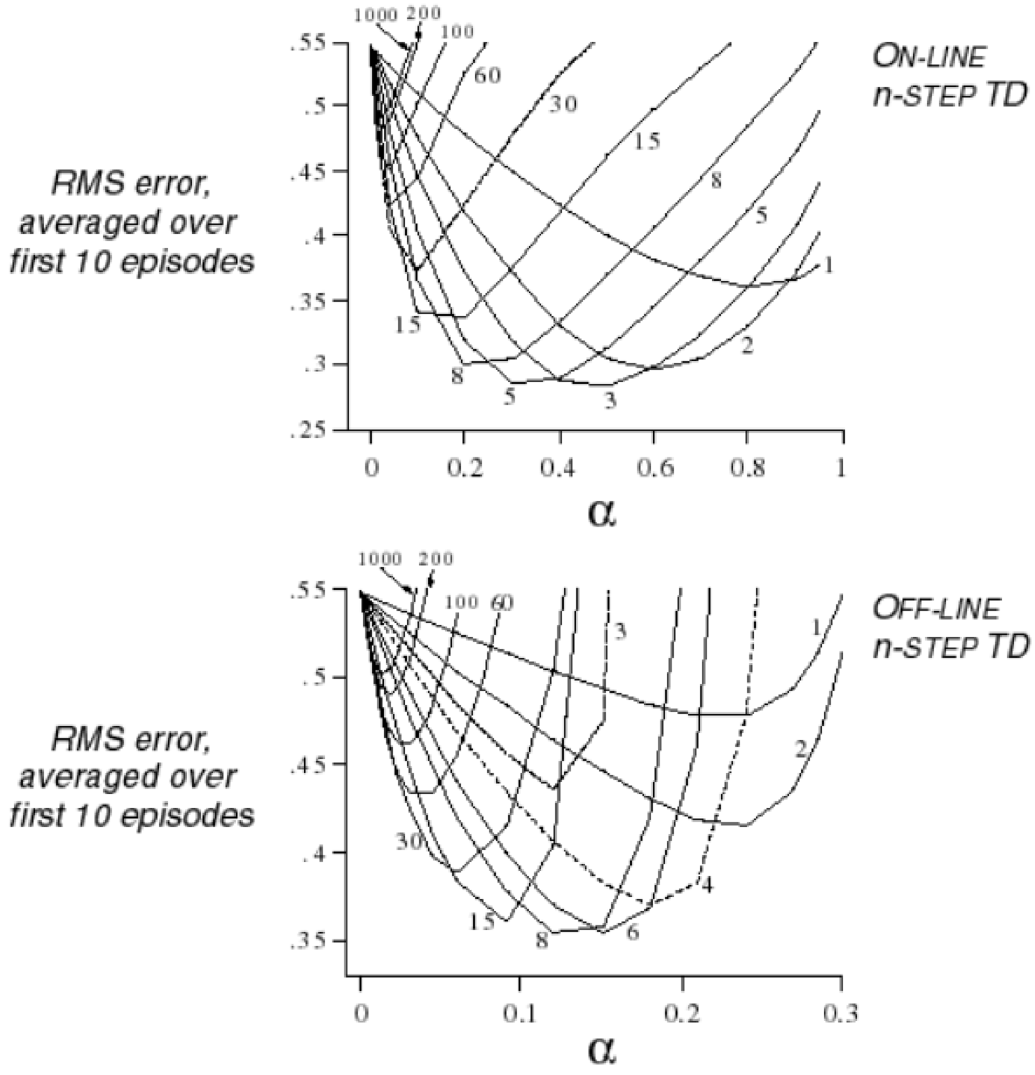


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Averaging *n*-Step Returns

- We can average *n*-step returns over different *n*
- e.g. average the 2-step and 4-step returns

$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$$

- Combines information from two different time-steps
- Can we efficiently combine information from all time-steps?

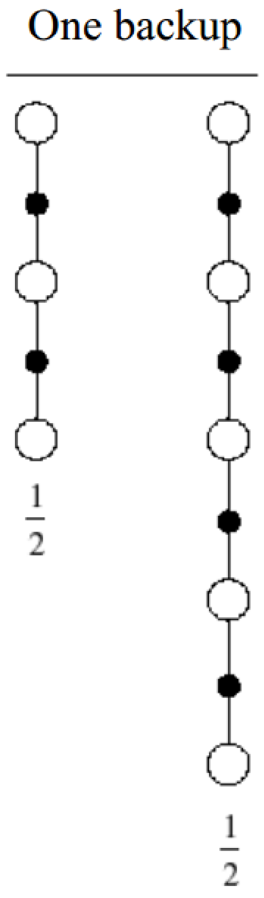


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

λ-return

- The λ*-return* G_t^λ combines all n-step returns G_t^n
- Using weight $(1-\lambda)\lambda^{n-1}$

$$G_t^\lambda = (1-\lambda)$$

- Forward-view TD(λ)

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

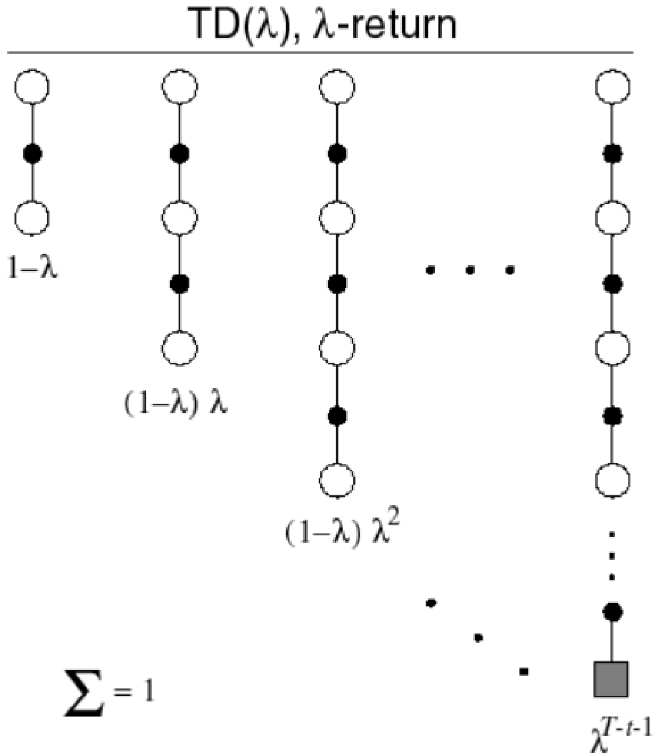
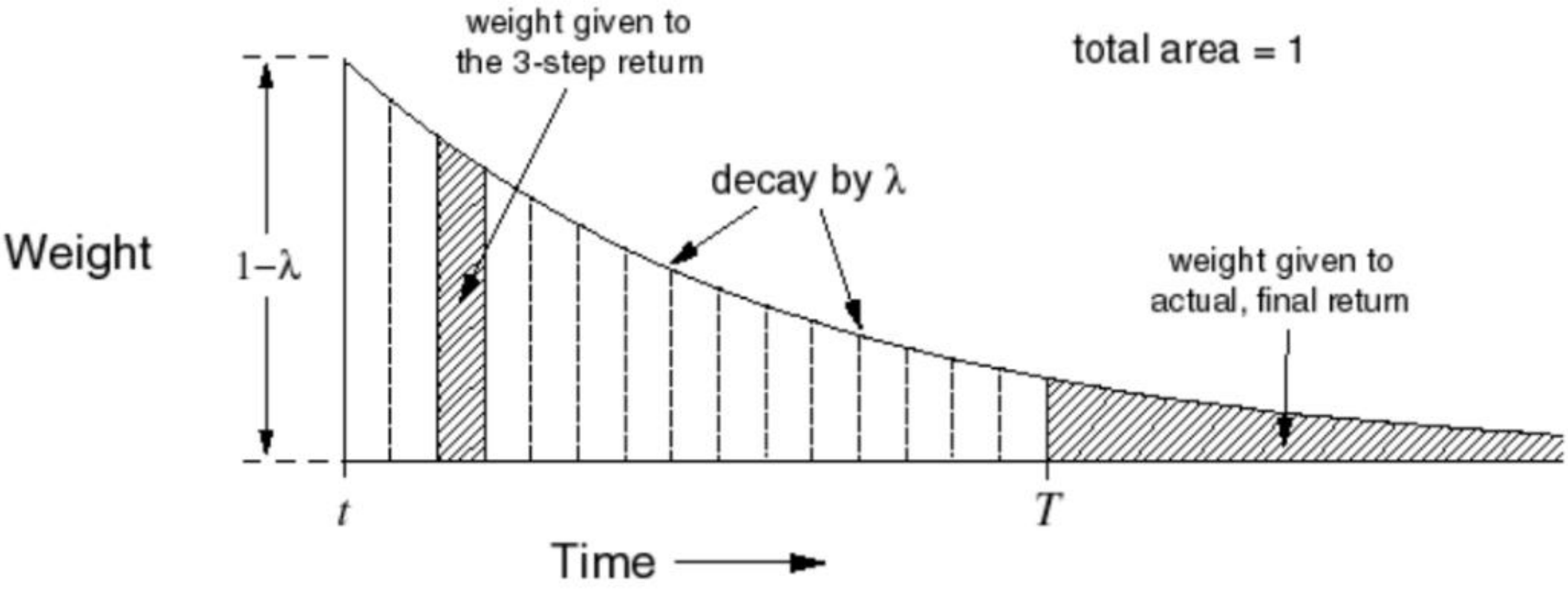


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

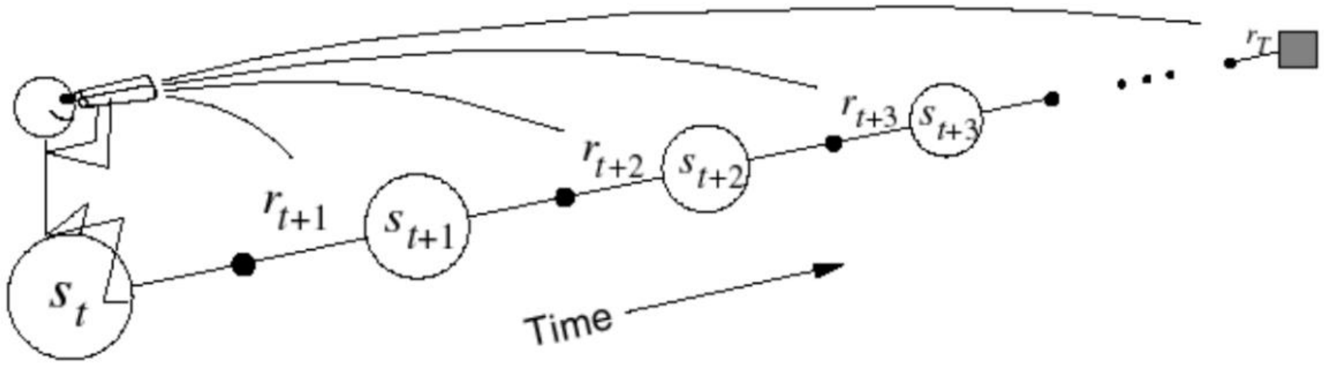
TD(λ)Weighting Function



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

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Forward-view TD(λ)



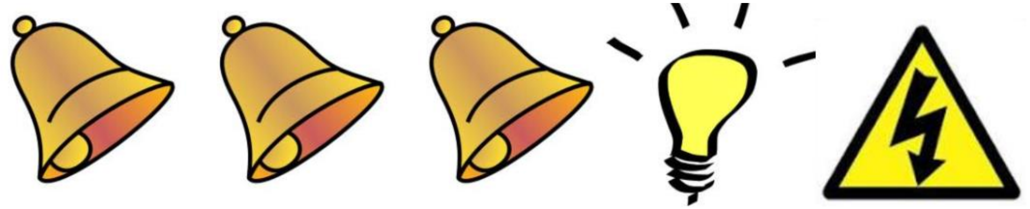
- Forward-view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Backward View TD(λ)

- Forward view provides theory
- Backward view provides mechanism
- Update online, every step, from incomplete sequences

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

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

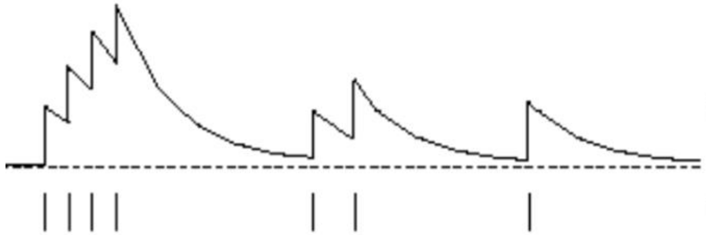


Image Credit: David Silver, Model-Free Prediction, UCL Course on RL

TD(λ) Algorithm

At each time step t in the rollout

current state

- Update eligibility trace for every state s

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

- Compute TD-error $\delta_t \; = \; R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
- Update value function $V(s)$ for every state s

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

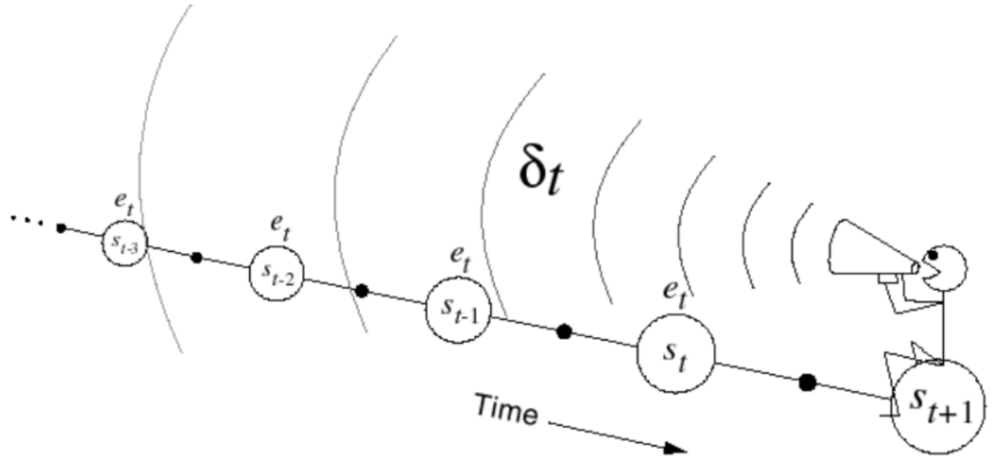


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

TD(λ) and TD(0)

- When $\lambda = \mathbf{0}$, only current state is updated

$$E_t(s) = \mathbf{1}(S_t = s)$$
$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

- This is exactly equivalent to TD(0) update

$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t$$

TD(λ) and MC

- TD(1) is roughly equivalent to MC
- Error is accumulated online, step-by-step
- If value function is only updated offline at end of episode
- Then total update is exactly the same as MC

Summary of Forward & Backward TD(λ)

- Batch Update
  - Updates are accumulated within episode, but applied in batch at the end of episode
  - → Backward TD(λ) is equivalent to Forward TD(λ)
- Online Update
  - TD(λ) updates are applied at each step within episode
  - Forward and backward-view TD(λ) are slightly different
  - Backward TD(λ) is typically more efficient
    - Analogy: SGD vs. batch GD

Temporal Difference Learning

Policy Optimization

Outline

- Introduction
- On-policy Montel Carlo control
- On-policy Temporal Difference control
- Off-policy Learning
- Summary

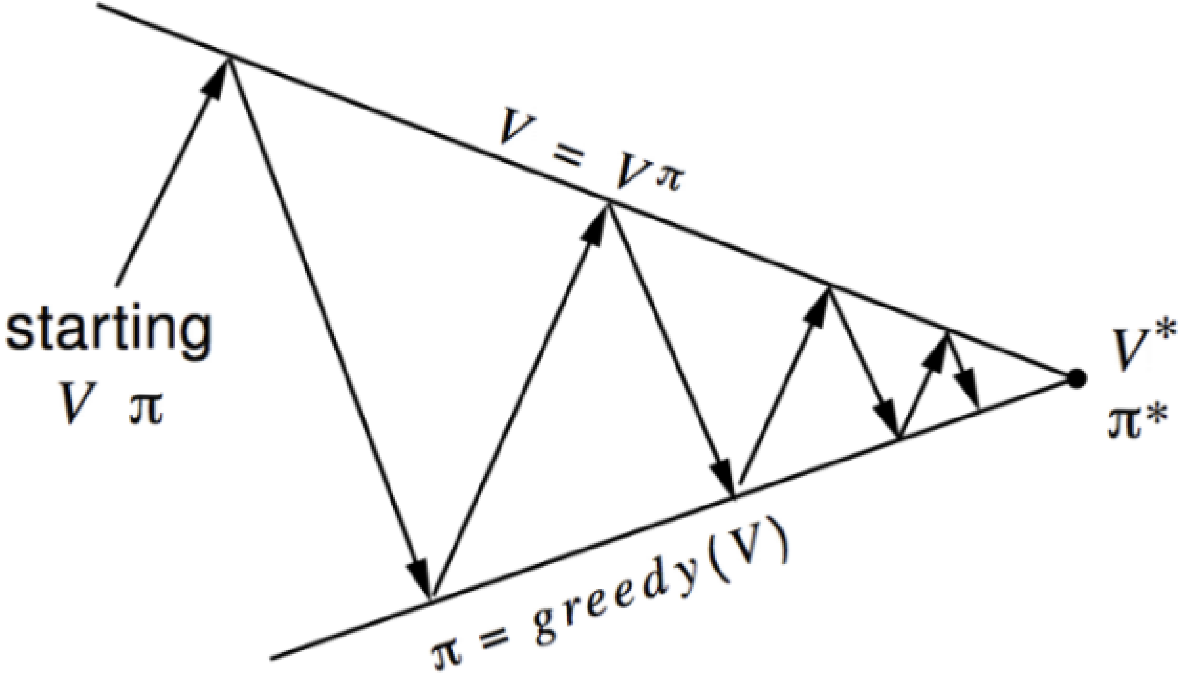
Introduction

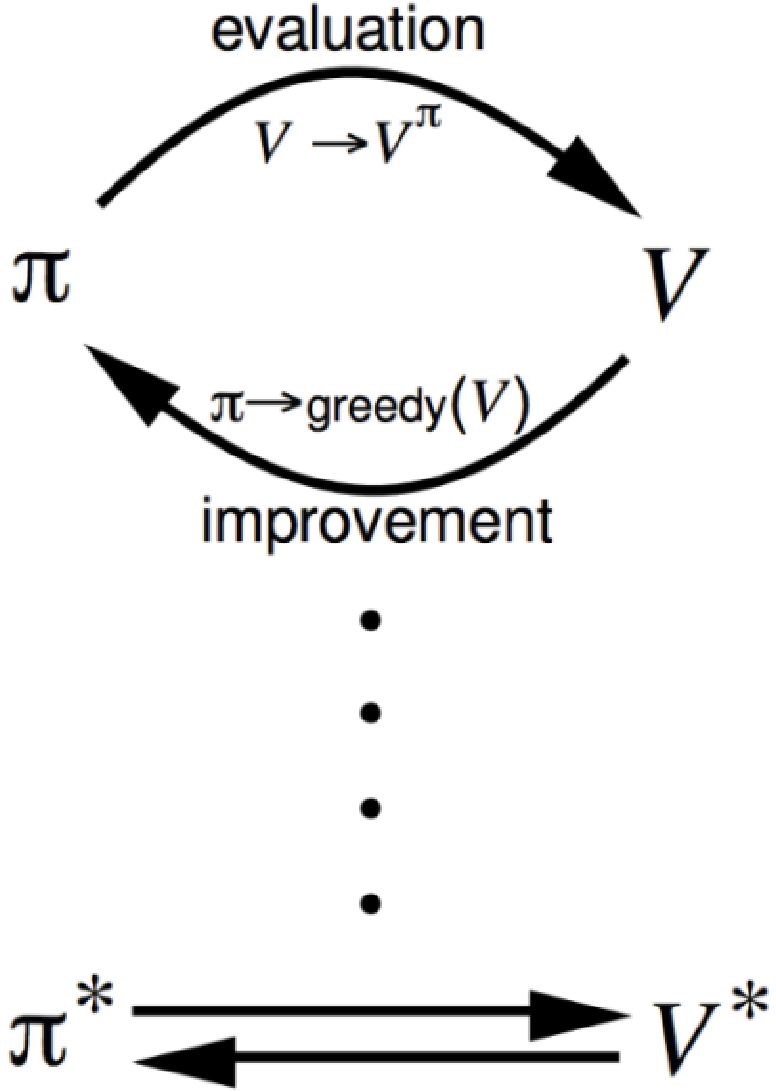
- Last lesson: model-free policy evaluation
  - Estimate the value function of an *unknown* MDP
- This lesson: model-free policy optimization
  - Optimize the value function of an *unknown* MDP

On and Off-Policy Learning

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

Generalised Policy Iteration (Refresher)





Policy evaluation Estimate v_π
e.g. Iterative policy evaluation

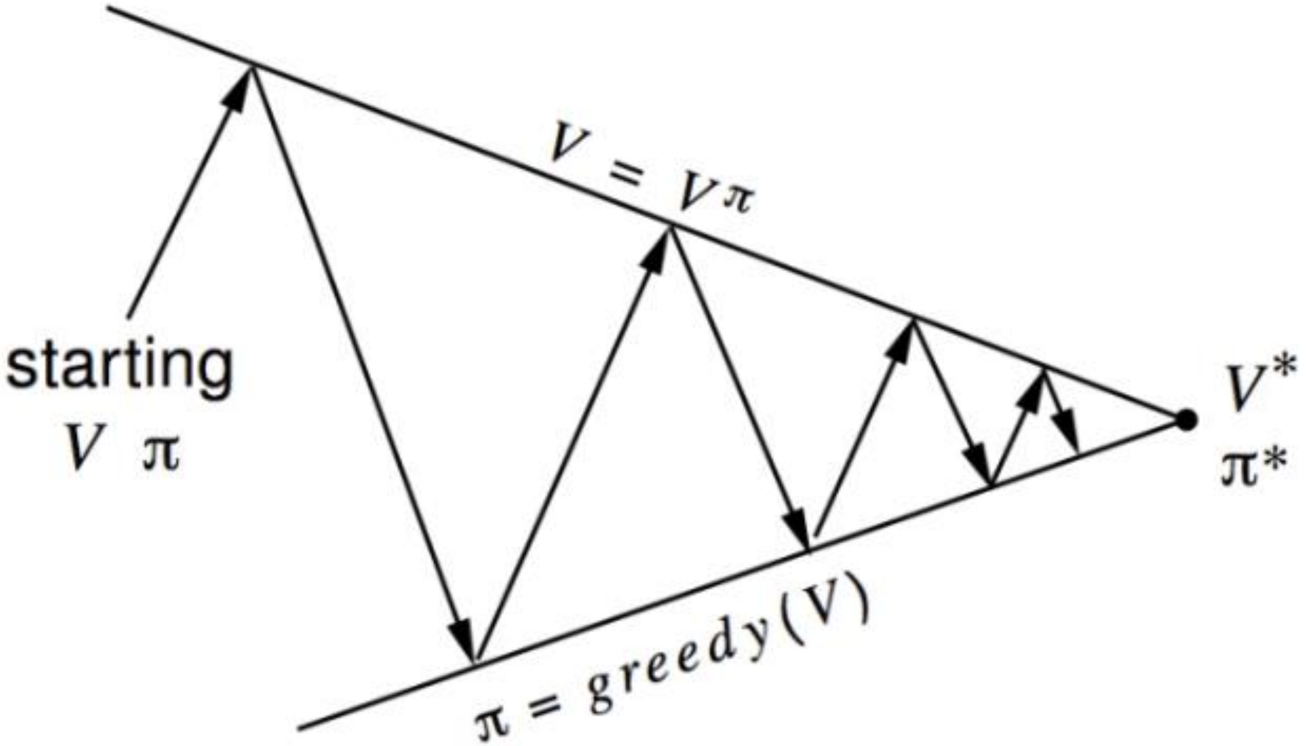
Policy improvement Generate $\pi' \geq \pi$
e.g. Greedy policy improvement

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Policy Optimization

On-Policy MC Control

Generalised Policy Iteration With Monte-Carlo Evaluation



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Model-Free Policy Iteration Using Action-Value Function

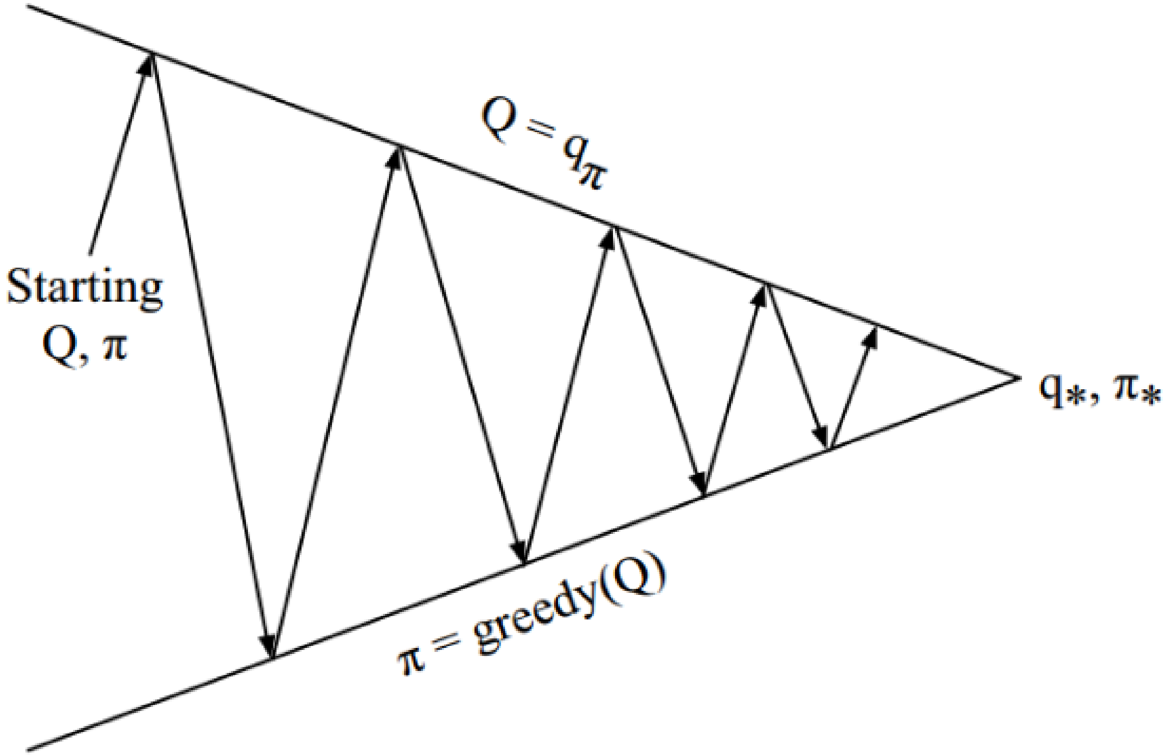
- Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; Q(s, a)$$

Generalised Policy Iteration with Action-Value Function



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Example of Greedy Action Selection

"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

- There are two doors in front of you.
- You open the left door and get reward 0 $V(\text{left}) = 0$
- You open the right door and get reward +1: $V(right) = +1$
- You open the right door and get reward +3: $V(right) = +3$
- You open the right door and get reward +2: $V(right) = +2$
- …
- Are you sure you've chosen the best door?

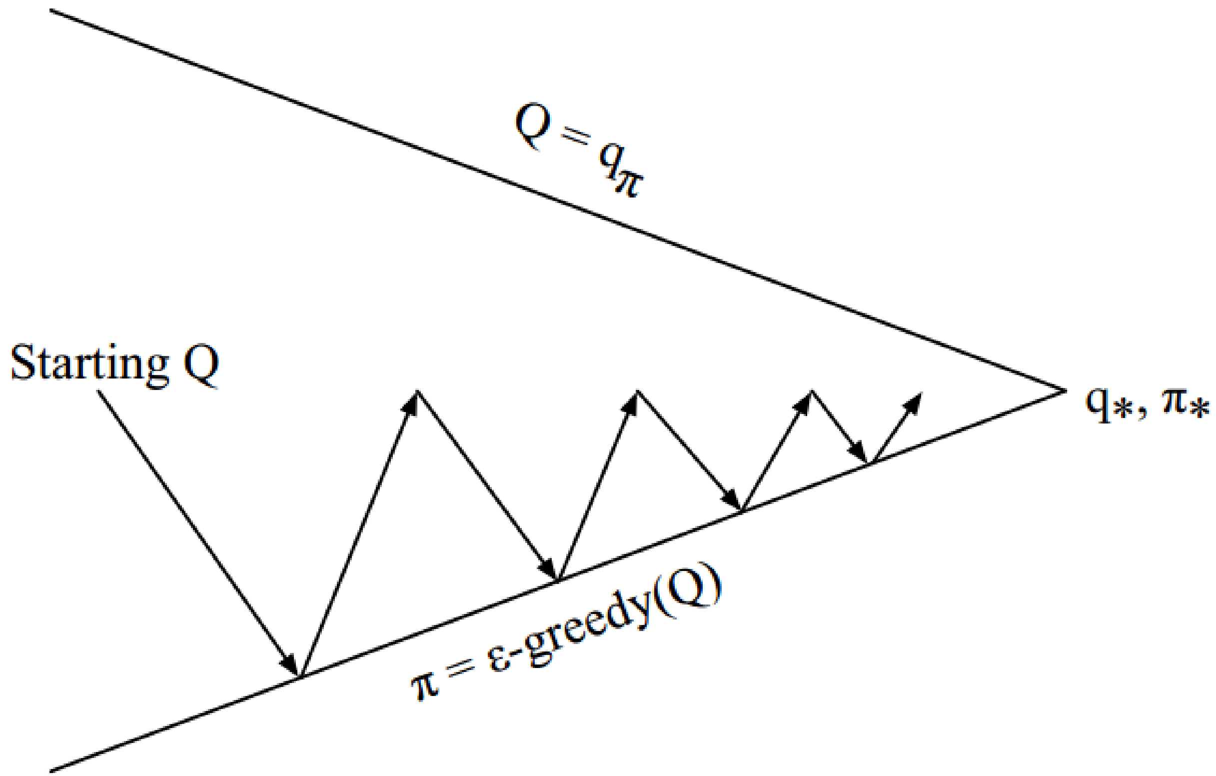
ϵ-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

- **Theorem**: *For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q_π is an improvement, i.e.* $v_{\pi'}(s) \geq v_\pi(s)$

Monte Carlo Control



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

GLIE

Definition: Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy

$$\lim_{k \to \infty} \pi_k(a|s) = \mathbf{1}(a = \operatorname*{argmax}_{a' \in A} Q_k(s, a'))$$

- Example: ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE Monte-Carlo Control

- Sample k^{th} episode using π: $\{S_1, A_1, R_2, \ldots, S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}\left(G_t - Q(S_t, A_t)\right)$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

- **Theorem**: GLIE Monte-Carlo control converges to the optimal action-value function

$$Q(s, a) \rightarrow q_*(s, a)$$

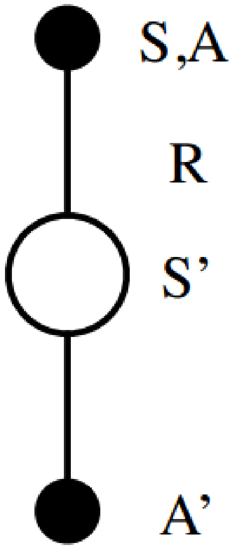
Policy Optimization

On-Policy TD Control

MC vs. TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy policy improvement
  - Update every time-step

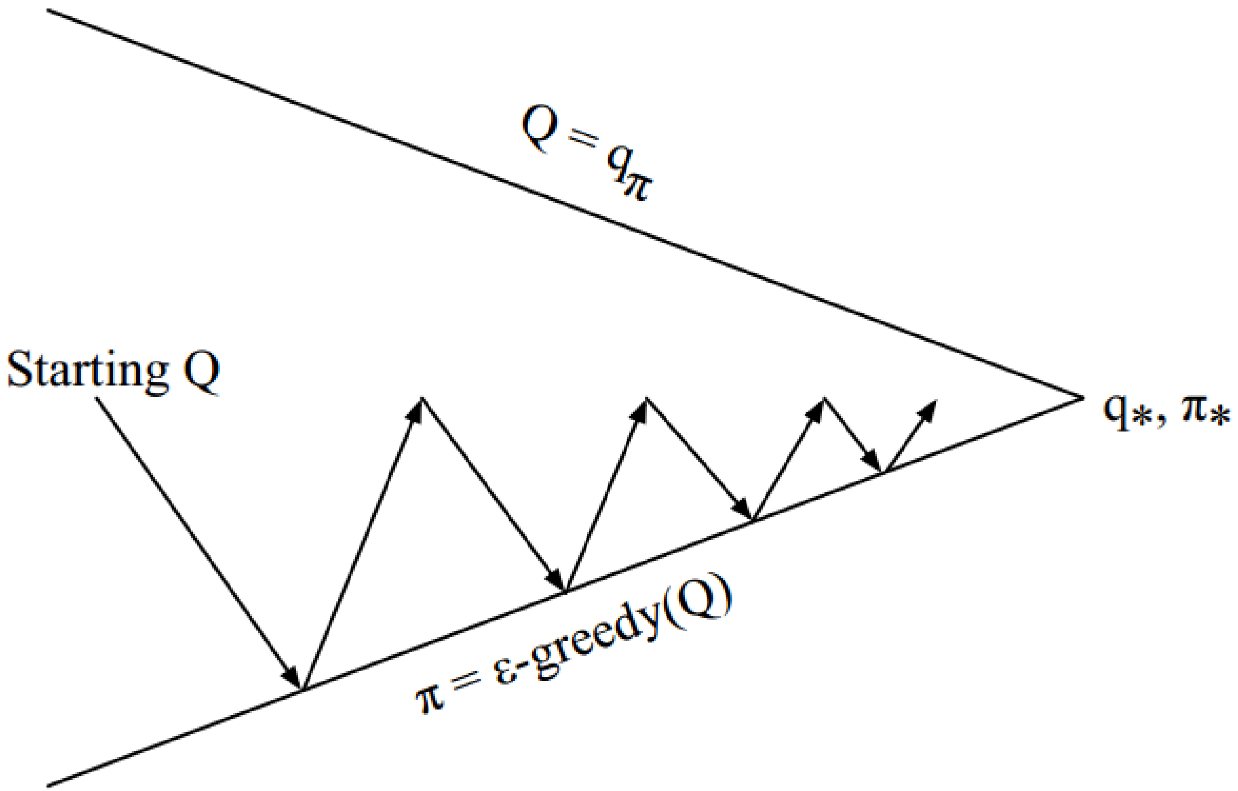
Sarsa for updating action-value function



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

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

On-Policy Control With Sarsa



Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

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

$$Q(s, a) \rightarrow q_*(s, a)$$

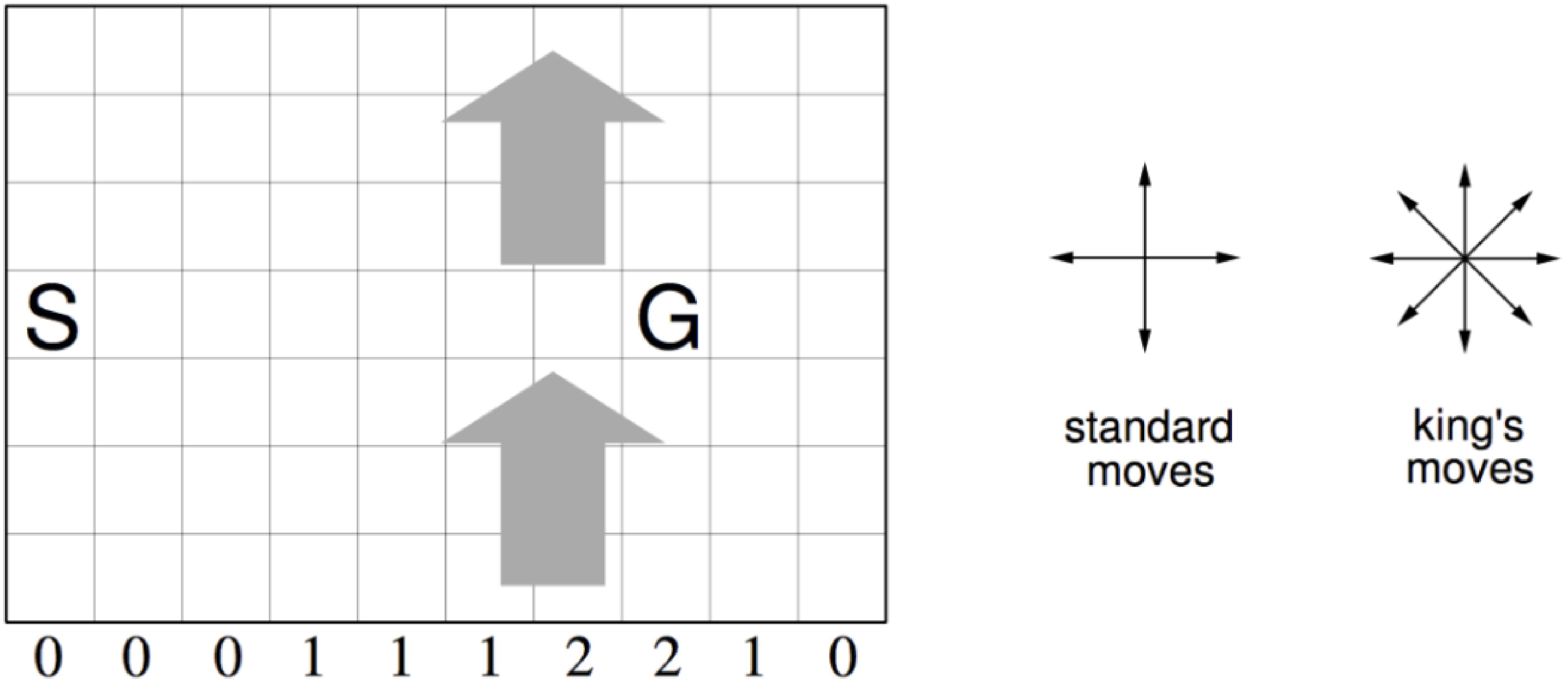
under the following conditions:

- GLIE sequence of policies $\pi_t(a|s)$
- Robbins-Monro sequence of step sizes α_t

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

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

Windy Gridworld Example



- Reward = -1 per time step until reaching goal
- Undiscounted

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Sarsa on the Windy Gridworld

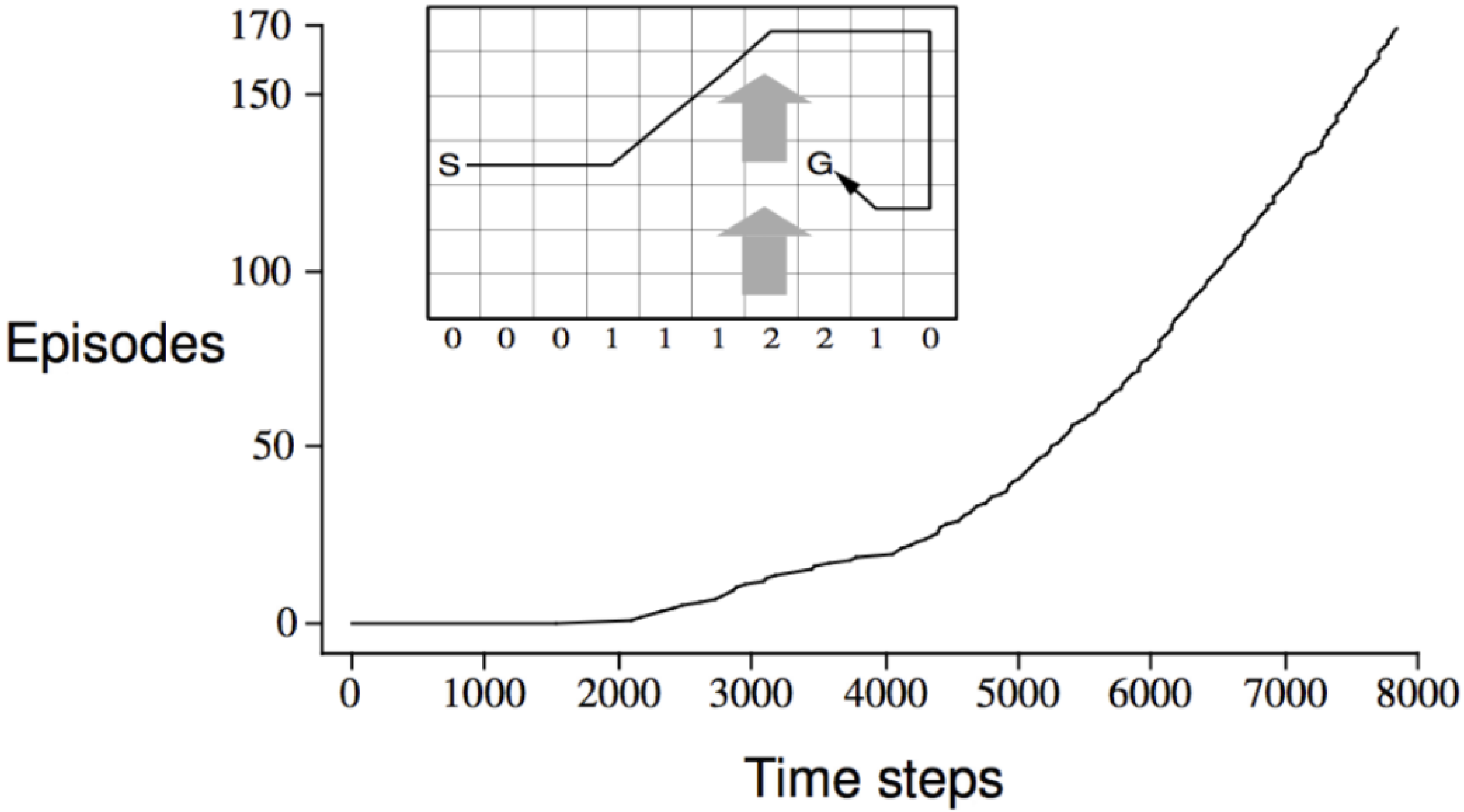


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Sarsa(λ)

n-Step Sarsa

- Consider the following n-step returns for $n = 1,2, \dots, \infty$:

$$
\begin{array}{lll}
n = 1 & (Sarsa) & q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \\
n = 2 & & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2}) \\
\dots & & \dots \\
n = \infty & (MC) & q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T
\end{array}
$$

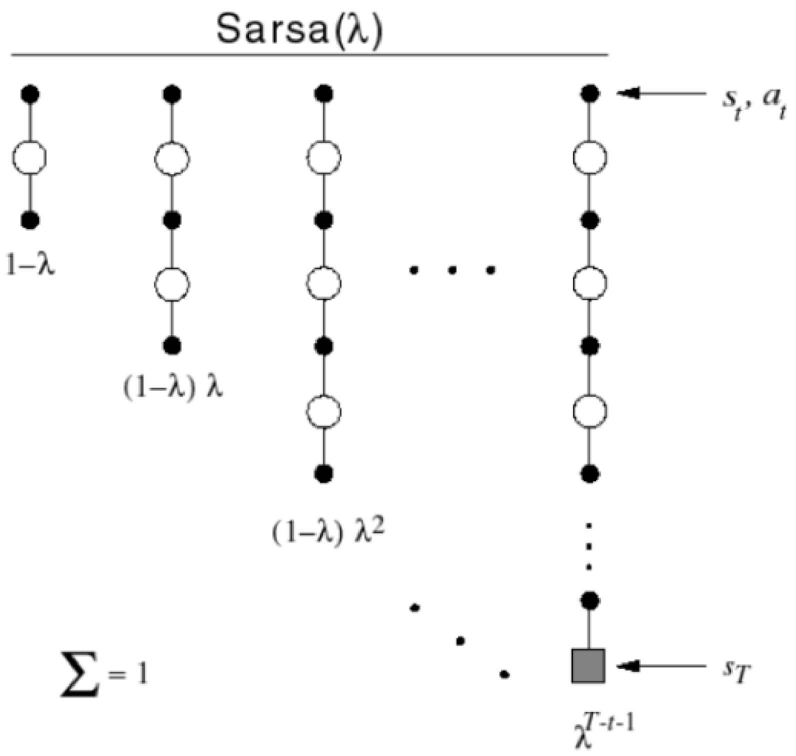
- Define the n-step Q-return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

- n-step Sarsa updates $Q(s, a)$ towards 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 Sarsa(λ)



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

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

- Forward-view Sarsa(λ)

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

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Backward View Sarsa(λ)

- Just like TD(λ), we use eligibility traces in an online algorithm
- But 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(λ) Algorithm

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

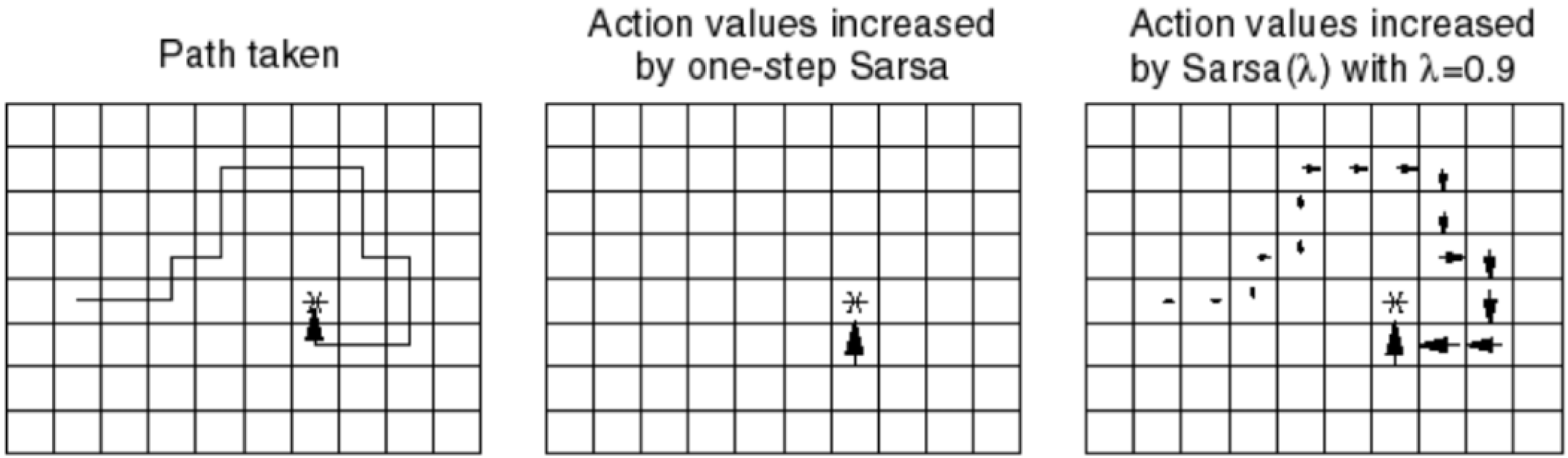


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Policy Optimization

Off-Policy Learning

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$
- While following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

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

Off-Policy Control with Q-Learning

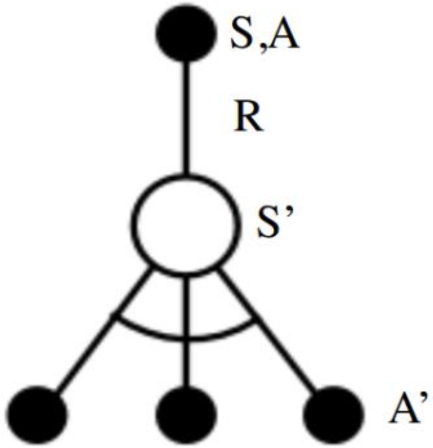
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target becomes:

$$R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$$

Q-Learning Control Algorithm



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

Theorem: *Q-learning control converges to the optimal action-value function*

$$Q(s,a) \rightarrow q_*(s,a)$$

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Q-Learning Algorithm

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'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

References

Richard Sutton. Reinforcement Learning: An Introduction (2nd edition).

David Silver's course on Reinforcement Learning:
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html