MONASH University
Information Technology

Acknowledgements

This material includes content adapted from instructional resources made available by David Silver as part of his Reinforcement Learning course at UCL under CC-BY-NC 4.0.

Refer to https://www.davidsilver.uk/teaching/ for full details.

MONASH University
Information Technology

MONASH University
Information Technology

FIT5226
Multi-agent System & Collective Behaviour

Wk 6: GLIE, SARSA and Q-Learning

Bernd Meyer, July 2022

From Model-based Prediction to Model-free Control

So far:

Model-based prediction

Estimate the value function of an *known* MDP (given P and R)

$$v'(s) \rightarrow \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{s,s'}^a v_\star^{s'}$$

The goal:

Model-free control

Optimise the value function of an *unknown* MDP

Step 1: From Model-based Prediction to Model-free-Prediction

To achieve model-free prediction (value estimation) we first need to get rid of the transition matrix that we use in model-based prediction to compute the expectation.

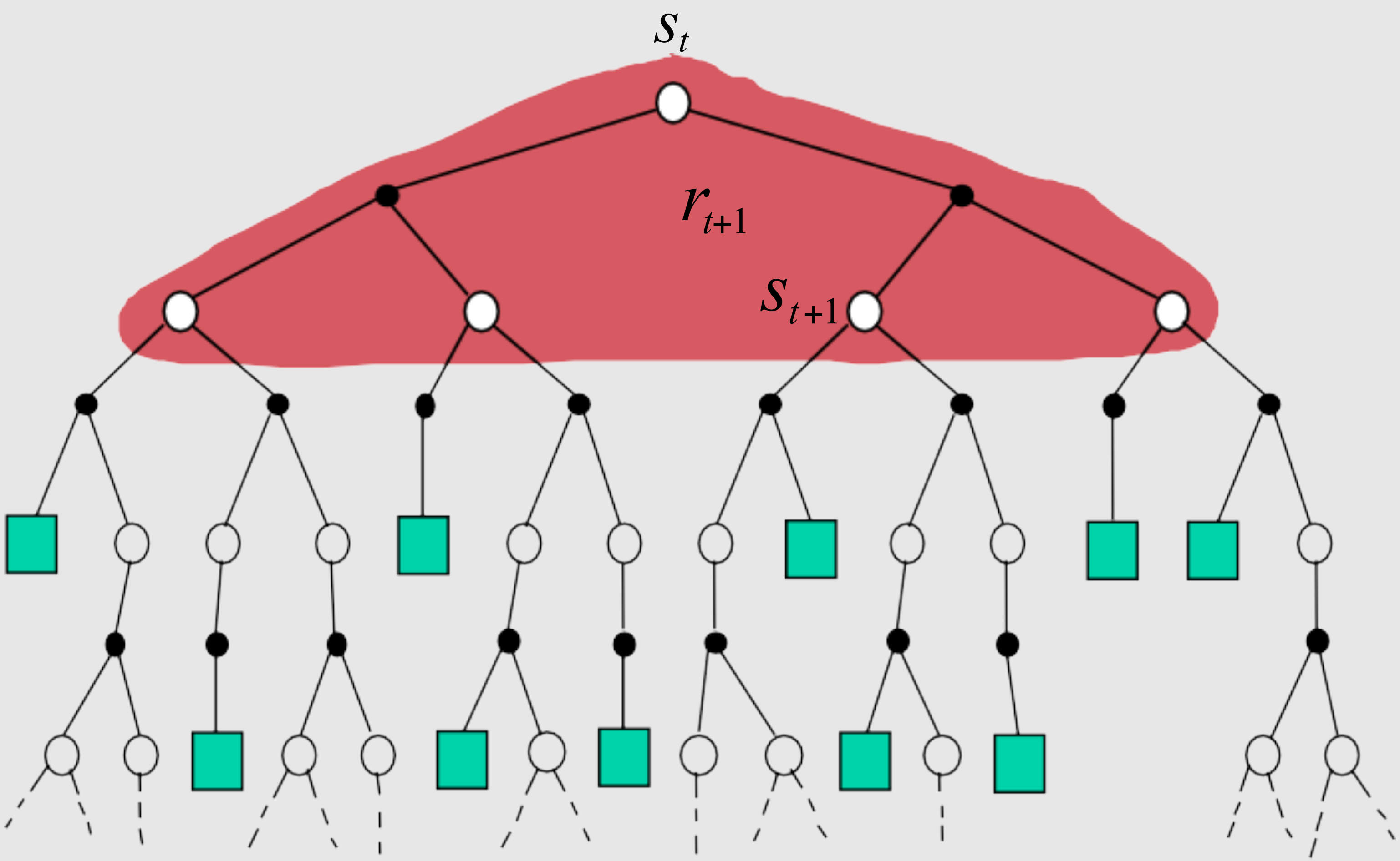
$$v'(s) \rightarrow \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{s,s'}^a v_\star^{s'}$$

We can use sampling instead.
In this way we will arrive at model-free estimation.

Sampling liberates us from needing a model

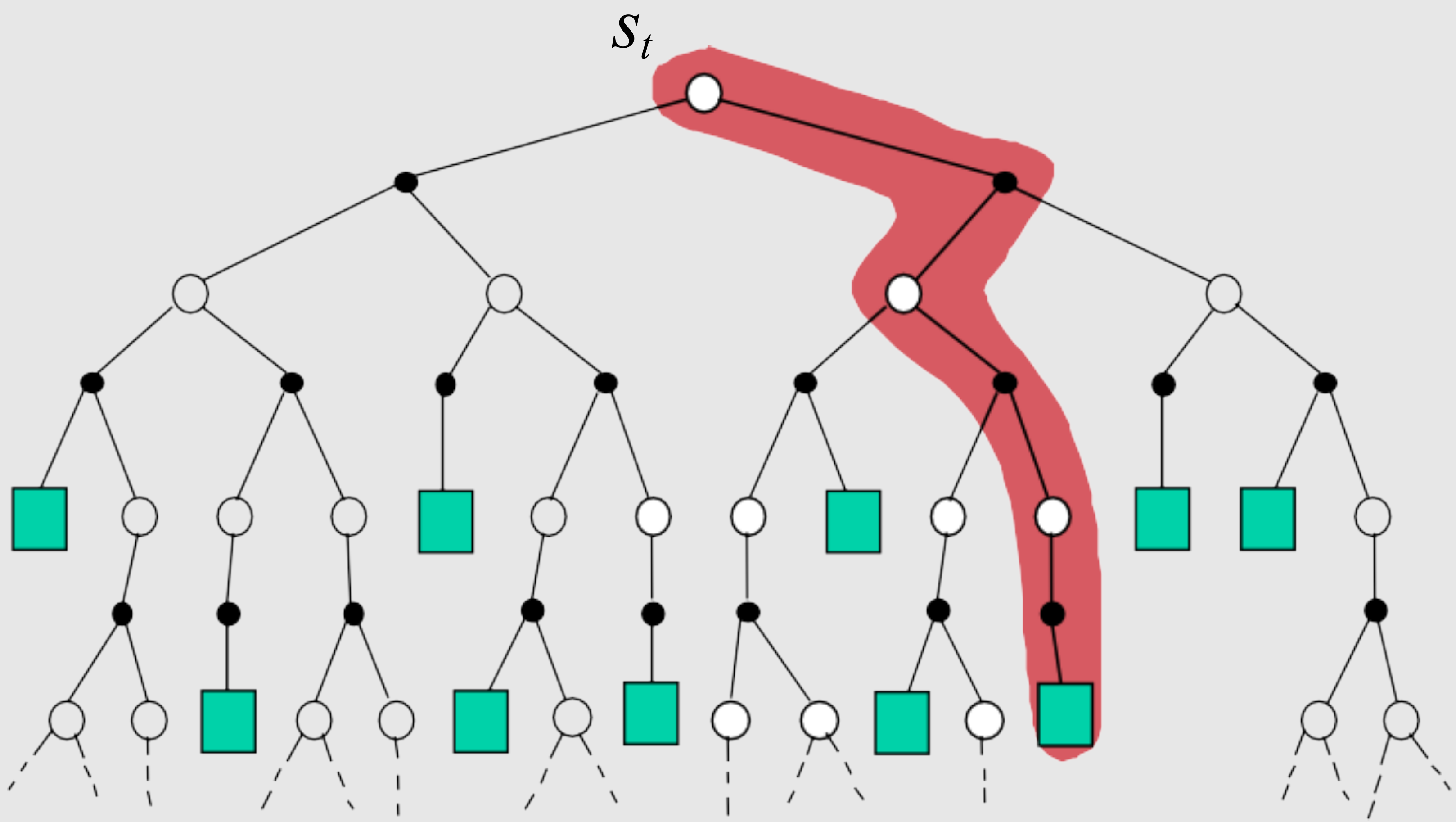
replace the expectation by an (incrementally computed) sample-mean

$$V(S_t) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})]$$



Monte-Carlo Update

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



RL Terminology: the structure of the update is called a “backup”

Monte-Carlo Policy Evaluation

Goal: learn v_π from episodes of experience under policy π

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

Recall that the return is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T$$

Recall that the value function is the expectation of the return:

$$v_\pi(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

Monte-Carlo policy evaluation uses empirical mean return instead of expected return

Every-Visit Monte-Carlo Policy Evaluation

To evaluate state s

- Every time-step 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 $V(s) \leftarrow S(s)/N(s)$
- $\lim_{N(s)\to\infty} V(s) = v_\pi(s)$

Incremental Monte-Carlo Updates

Update the empirical mean of $V(s)$ incrementally after episode $S_1, A_1, R_2, \ldots, S_T$

For each state S_t with return G_t

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\Big((G_t - V(S_t)\Big)$$

In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes:

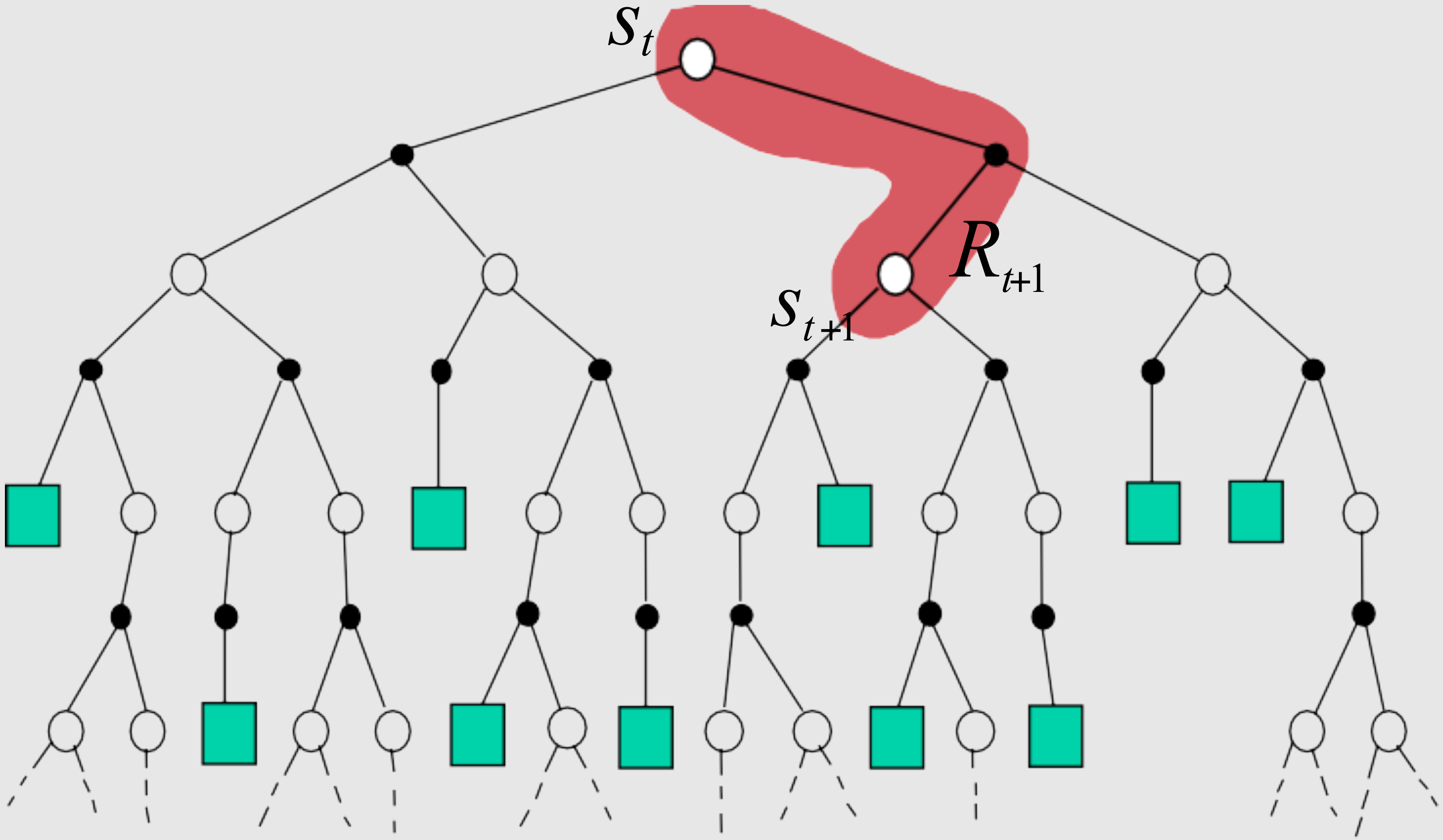
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Monte-Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions or rewards
- MC learns from complete episodes: no bootstrapping
- MC uses the simplest possible idea: value = sample mean return
- **Caveat:**
  - All episodes must terminate
  - **MC can only be applied to episodic MDPs**

Temporal-Difference Update

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



Monte-Carlo versus Temporal Difference

Goal: learn V_π online from experience under policy π

- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t

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

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

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

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

Advantages and Disadvantages of MC vs. TD

- TD can learn before and without the final outcome

- TD can learn online after every step
- MC must wait until end of episode before return is known

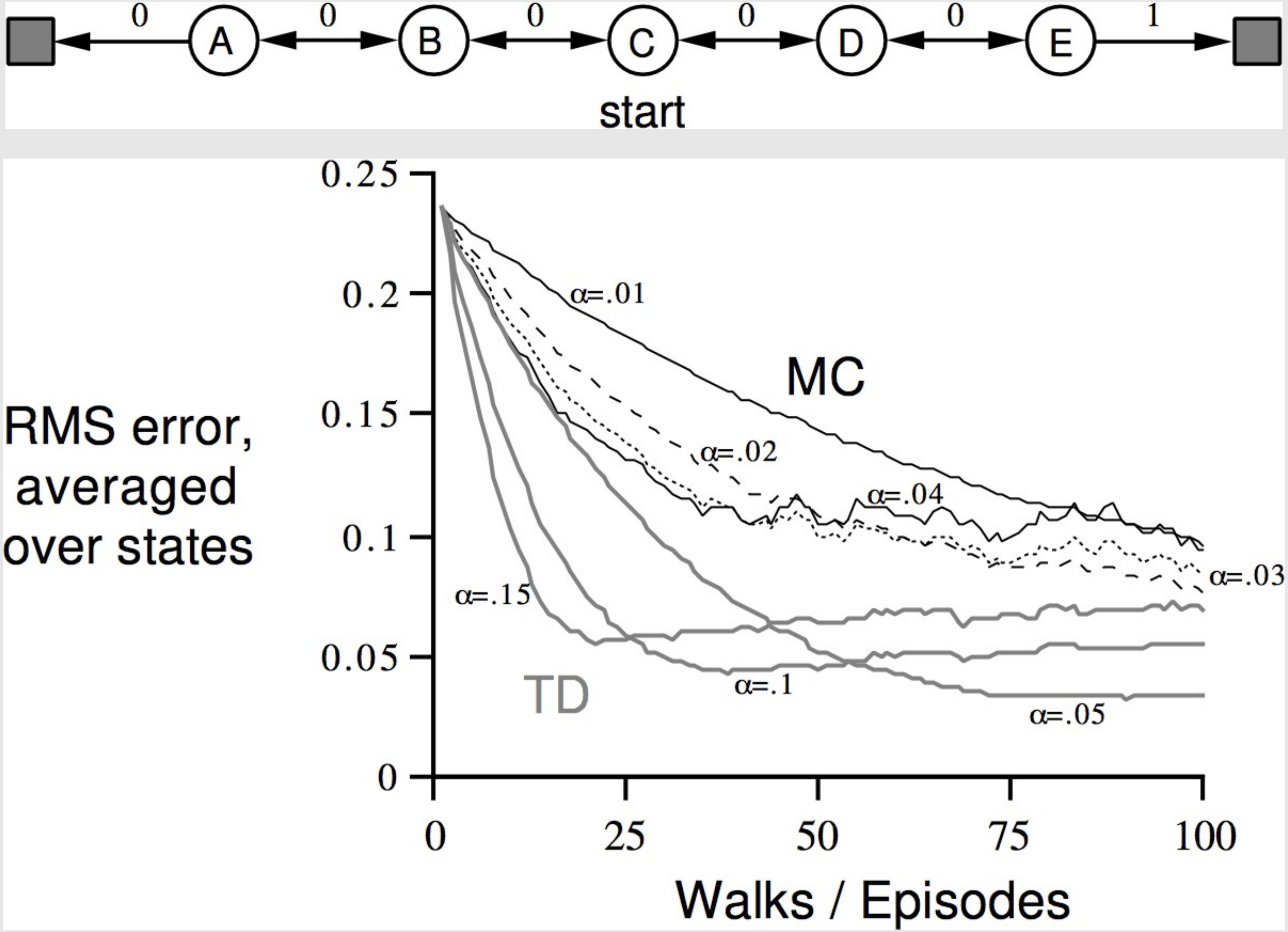
- TD can learn from incomplete sequences MC can only learn from complete sequences
- TD works in continuing (non-terminating) environments
- MC only works for episodic (terminating) environments

Advantages and Disadvantages of MC vs. TD (cont'd)

- MC has high variance, zero bias
  - good convergence properties
  - not very sensitive to initial value
  - very simple to understand and use

- TD has low variance, some bias
  - usually more efficient than MC
  - TD converges to $v_\pi(s)$ (with value tables)
  - more sensitive to initial value

Random Walk: MC vs. TD



Step 2: From Model-free Prediction to Control

Ultimately, we need to find the optimal policy π
So far we have worked with the state-value function $V(s)$

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

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\ R_s^a + P_{s,s'}^a V(s')$$

We can instead directly work with the action-value function.

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

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

But now we need to find Q instead of V !

Monte-Carlo Policy Iteration (simple)

Policy evaluation Monte-Carlo policy evaluation (as above), $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

with probability $(1 - \epsilon)$ choose $\underset{a \in A}{\operatorname{argmax}}\ Q(S, a)$;

with probability ϵ choose randomly

There are alternative ways to handle the policy improvement but they must be **GLIE**

GLIE Monte-Carlo Control (Greedy in the limit with infinite exploration)

Sample k-th episode using current π : $\{S_1, A_1, R_2, \ldots, R_T\} \sim \pi$

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

$\epsilon \leftarrow 1/k$ ⟵ Infinite Exploration

$\pi \leftarrow \epsilon\text{–greedy}(Q)$ ⟵ Greedy in the limit

Theorem

GLIE Monte-Carlo control converges to the optimal action-value function $Q(s, a) \rightarrow q_\star(s, a)$

MC vs. TD for Control

Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)

- can use incomplete sequences
- lower variance

Thus we expect improvement by using TD instead of MC in the control loop

- Apply TD to Q(S, A)
- ϵ-greedy policy improvement
- update every time-step

Policy Iteration with Sarsa

Every time step

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

Policy improvement ϵ-greedy policy improvement

with probability $(1 - \epsilon)$ choose $\operatorname*{argmax}_{a \in A} Q(S, a)$;

with probability ϵ choose randomly

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

Sarsa Algorithm for On-Policy Control

Every time-step

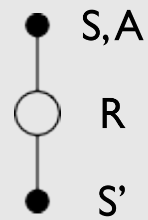
- Policy evaluation Sarsa $Q \approx q_\pi$
- Policy improvement ϵ-greedy policy improvement

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

Summary: SARSA - TD Backups

We now only consider sample TD-backups

Using sample rewards and sample transitions



(S, A, R, S)

We do not need the reward function R and transition dynamics P

Advantages:

- Model-free: no advance knowledge of MDP required
- Breaks the curse of dimensionality through sampling
- Cost of an individual backup is constant, independent of n = |S|

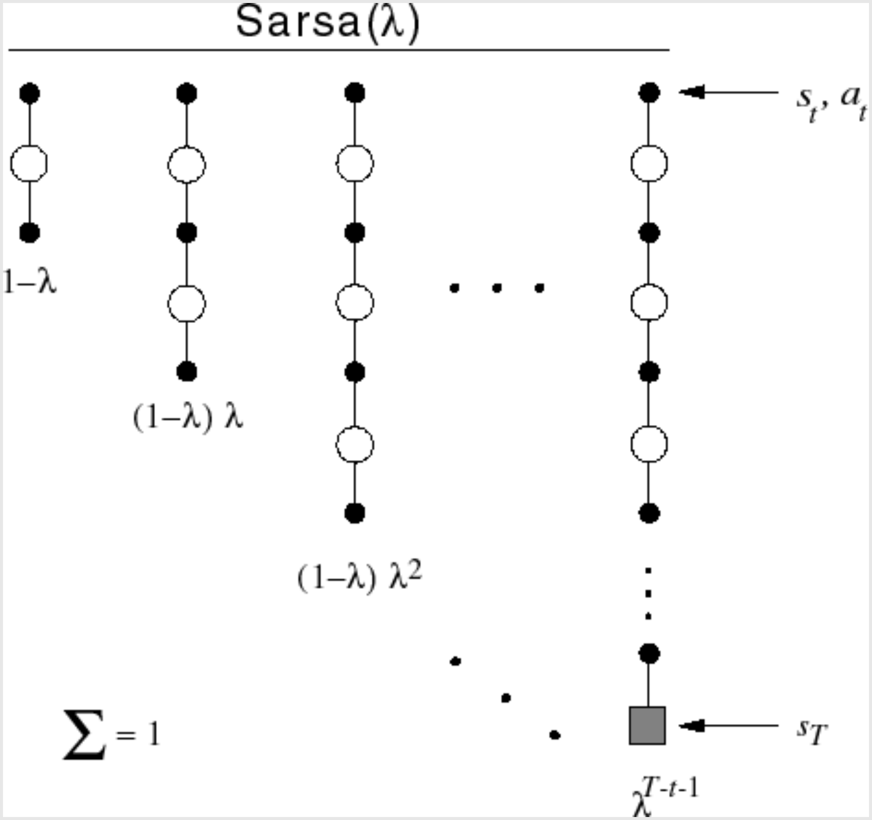
Convergence of SARSA

Theorem

Sarsa converges to the optimal action-value function $Q(s,a) \rightarrow q_\star(s,a)$, under the following conditions:

- GLIE sequence of policies $\pi_t(a \mid s)$
- Robbins-Monro sequence of step-sizes α_t
  - $\sum_{t=1}^{\infty} \alpha_t = \infty$
  - $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

SARSA(λ) and n-step SARSA



n=1 Sarsa $q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1})$

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

…

n= ∞ MC $q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$

n-step Sarsa

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

Sarsa(λ) combines all *n-step* q-returns in the episode

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

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

SARSA(λ) Gridworld Example [Sutton Barto]

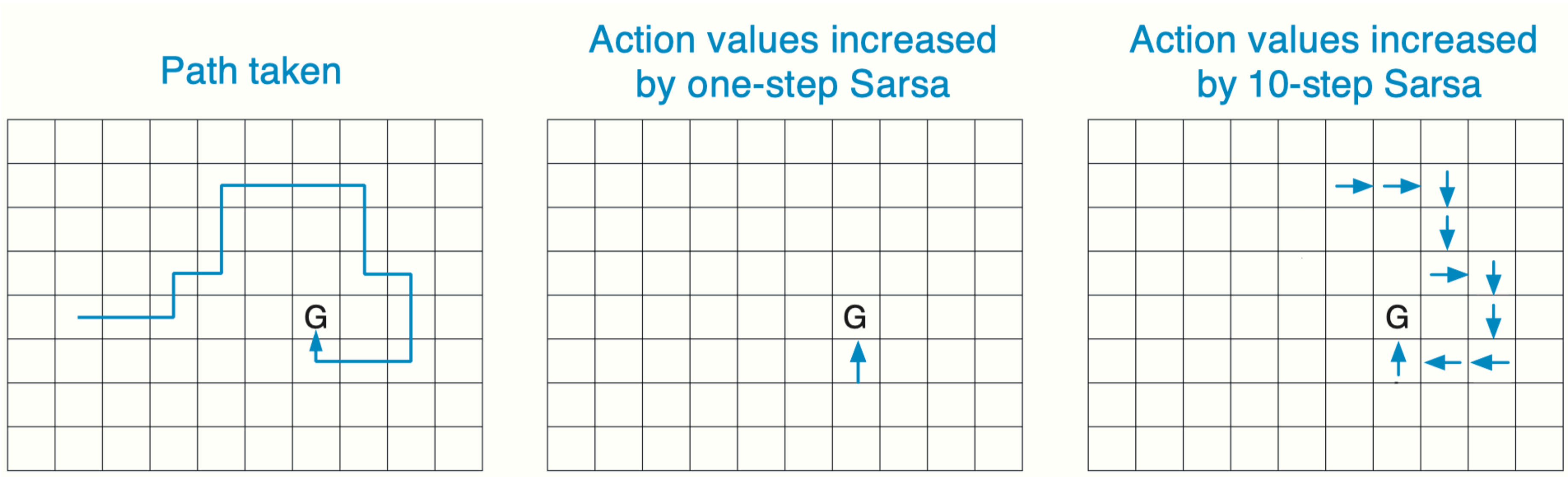


Figure 7.4: Gridworld example of the speedup of policy learning due to the use of n-step methods. The first panel shows the path taken by an agent in a single episode, ending at a location of high reward, marked by the G. In this example the values were all initially 0, and all rewards were zero except for a positive reward at G. The arrows in the other two panels show which action values were strengthened as a result of this path by one-step and n-step Sarsa methods. The one-step method strengthens only the last action of the sequence of actions that led to the high reward, whereas the n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode.

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

Backward View TD(λ)

- Forward view (above) provides theory
- Backward view provides implementation mechanism
  - updates online, every step,
    from incomplete sequences

On-Policy vs 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

Off-Policy Learning

Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
while following behaviour policy $\mu(a \mid s)$
$\{S_1, A_1, R_2, \ldots, S_T\} \quad \sim \mu$

Why is this important or useful?

- **Learn about optimal policy following *exploratory* policy**
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$

Q-Learning

We now consider off-policy learning of action-values Q(s, a)

- the next action is chosen using ***behaviour*** policy

  $A_t \sim \mu(\cdot \mid S_t)$ *typically ϵ-greedy w.r.t. Q*

  A_t determines the next state S_{t+1} and reward R_{t+1}

- we then consider a successor action from the ***target*** policy

  $A^\pi \sim \pi(\cdot \mid S_{t+1})$ *typically greedy w.r.t. Q*

  we determine the state value of this next state S_{t+1} from its *Q*-values by choosing an action A^π according to the *target* policy

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A^\pi) - Q(S_t, A_t)\right)$$

Off-Policy Control with Q-Learning

We now improve both behaviour and target policies.
The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(S, a)$.

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 *Q*-learning target then simplifies:

$$\begin{aligned}
& R_{t+1} + \gamma Q(S_{t+1}, A^{\pi}) \\
= \quad & R_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\operatorname{argmax}} \; Q(S_{t+1}, a')) \\
= \quad & R_{t+1} + \gamma \max_{a'} \; Q(S_{t+1}, a')
\end{aligned}$$

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

Q-Learning Control

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

Theorem

Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow q_{\star}(s,a)$

Q-Learning Algorithm for Off-Policy Control

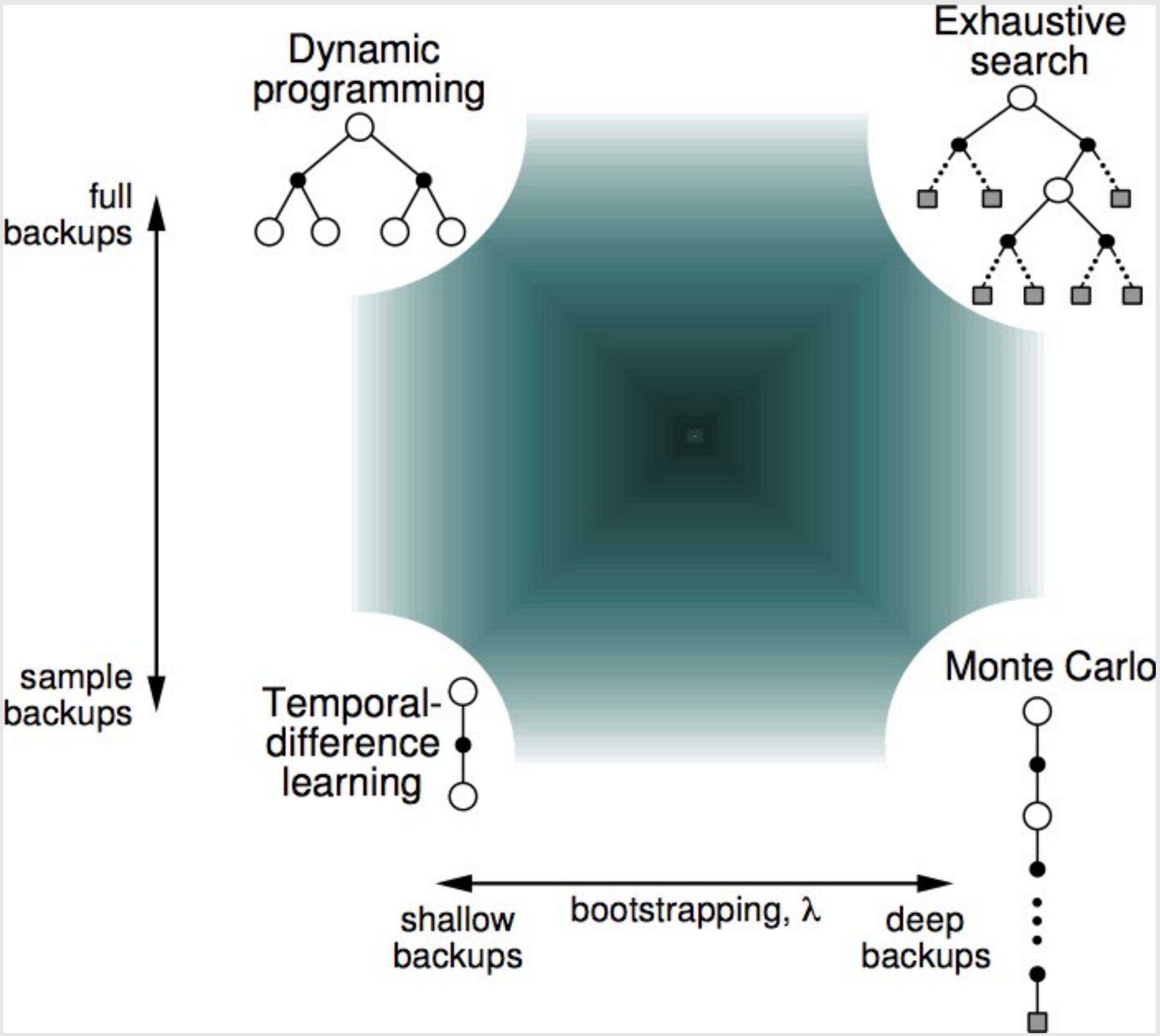
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, 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

Relationship Between DP and TD

	Full Backup (DP)	*Sample Backup (TD)*	*Update*
Bellman Expectation Equation for $v_\pi(s)$	Policy Evaluation	TD Learning	$V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Bellman Expectation Equation for $q_\pi(s, a)$	Policy Iteration	Sarsa	$Q(S, A) \xleftarrow{\alpha} R + \gamma Q(S', A')$
Bellman Optimality Equation for $q_*(s, a)$		Q-Learning	$Q(S, A) \xleftarrow{\alpha} R + \gamma \max_{a' \in A} Q(S', a')$

where $x \xleftarrow{\alpha} y$ is $x \leftarrow x + \alpha(y - x)$

Unified View of Reinforcement Learning



Take home lessons

- To work without a model with resort to sampling
- We can either learn from samples of full episodes (MC) or at every step (TD)
- MC methods have higher variance but no bias.
- TD is usually more efficient but sensitive to initial values
- To guarantee convergence sampling (usually) needs to be GLIE
- On-policy methods use the same policy they are learning to sample; off-policy methods can use a sampling policy that is different from the one being learned
- SARSA and Q-Learning are simple TD methods to learn the q function
- SARSA is on-policy; Q-Learning is off-policy