Value Based - Part 2

Deep Reinforcement Learning

Mohammad Hossein Rohban, Ph.D.

Courtesy: Some slides are adopted from CS 285 Berkeley, and CS 234 Stanford, and Pieter Abbeel's compact series on RL.

RIML Lab, CE Department, Sharif University of Technology

Spring 2026

Disadvantages of Monte-Carlo Learning

- We have seen MC algorithms can be used to learn value predictions
- But when episodes are long, learning can be slow
  - ☑ we have to wait until an episode ends before we can learn…
  - ☑ return can have high variance
    - Which one is more? First-visit or every-visit
- Are there alternatives? (Spoiler: yes)

Monte Carlo Control

Monte-Carlo Control

Algorithm:

Repeat:

Sample episode $1, \dots, k, \dots,$ **using** $\pi: \{S_1, A_1, R_2, \dots, S_T\} \sim \pi$

For each state S_t and action A_t in the episode:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t \big(G_t - q(S_t, A_t)\big)$$

e.g.,

$$\alpha_t = \frac{1}{N(S_t, A_t)} \quad of \quad \alpha_t = 1/k$$

Improve policy based on new action-value function

$$\pi^{new}(s) = \underset{a \in A}{\operatorname{argmax}}\, q(s, a)$$

Any issue?

- Let's consider this example:
- Discount = 1, start in state H.

A	B	C	D	E	F	G	H	I	J
r=10	→	→	→	→	→	↑	→	→	r=1

Epsilon Greedy Policy

- Simple idea to balance exploration and achieving rewards
- Let $|A|$ be the number of actions
- Then an ϵ-greedy policy w.r.t a state action value $Q(s,a)$ is

$$\pi(a|s) =$$

- ☑ $\underset{a}{\mathrm{arg}max} Q(s,a),\ w.\mathrm{P}rob\ 1-\epsilon+\frac{\epsilon}{|A|}$
- ☑ $a' \neq \mathrm{arg}max Q(s,a)\, w.\mathrm{P}rob\ \frac{\epsilon}{|A|}$

Does this hurt improvement?

Theorem

For any ϵ-greedy policy π_i, the ϵ-greedy policy w.r.t. Q^{π_i}, π_{i+1} is a monotonic improvement $V^{\pi_{i+1}} \geq V^{\pi_i}$

Monte-Carlo Control (done right)

Algorithm:

Repeat:

Sample episode $1, \dots, k, \dots,$ **using** $\pi: \{S_1, A_1, R_2, \dots, S_T\} \sim \pi$

For each state S_t and action A_t in the episode:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t (G_t - q(S_t, A_t))$$

e.g.,

$$\alpha_t = \frac{1}{N(S_t, A_t)} \quad of \quad \alpha_t = 1/k$$

Improve policy based on new action-value function

$$\pi(a|s) =$$

$$\underset{a}{\operatorname{argmax}} Q(s,a), w.\text{P}rob \;\; 1 - \epsilon + \frac{\epsilon}{|A|}$$

$$a' \neq \operatorname{argmax} Q(s,a)\, w.\text{P}rob \;\; \frac{\epsilon}{|A|}$$

Disadvantages of MC Learning

- We have seen MC algorithms can be used to learn value predictions
- But when episodes are long, learning can be slow
  - ☑ ...we have to wait until an episode ends before we can learn
  - ☑ ...return can have high variance
- Are there alternatives? (Yes)

Temporal Difference Learning

Prediction

TD Overview

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes, by bootstrapping
- TD updates a guess towards a guess

TD Learning by Sampling Bellman Equations

- Bellman update equations:

$$v_{k+1}(s) = \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)]$$

- We can sample this!

$$v_{t+1}(S_t) = R_{t+1} + \gamma v_t(S_{t+1})$$

- Samples could be averaged, in a similar way to MC:

$$v_{t+1}(S_t) \leftarrow v_t(S_t) + \alpha \left(\underbrace{\underbrace{R_{t+1} + \gamma v_t(S_{t+1})}_{\text{target}} - v_t(S_t)}_{\text{temporal difference error } \delta_t} \right)$$

Temporal Difference Learning

- Prediction setting: learn v_π online from experience under policy π
- Monte Carlo
  - Update value $v_n(S_t)$ towards sampled return G_t

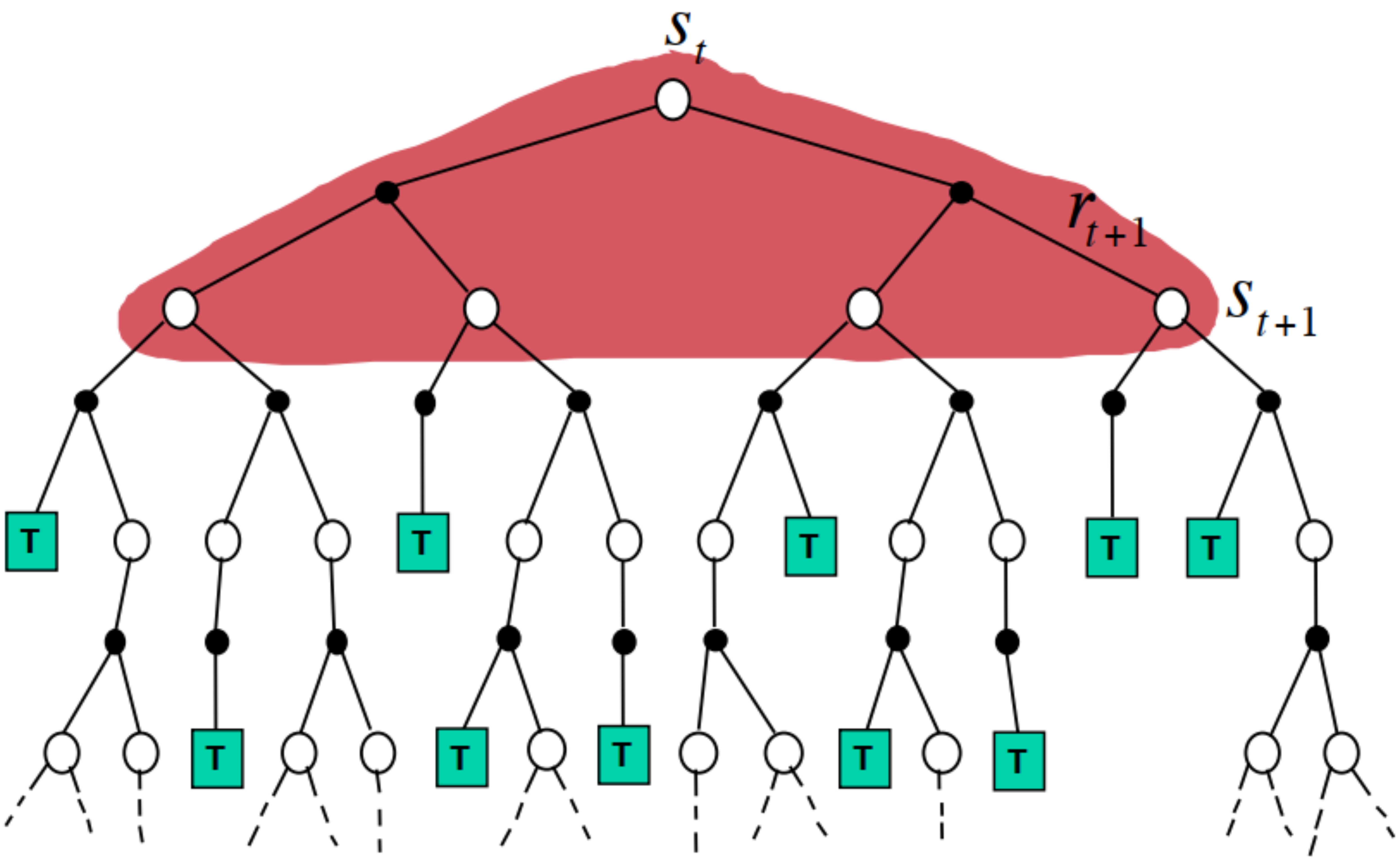
$$v_{n+1}(S_t) = v_n(S_t) + \alpha(G_t - v_n(S_t))$$

- TD Learning
  - Update value $V_t(S_t)$ towards estimated return $R_{t+1} + \gamma v(S_{t+1})$

$$v_{t+1}(S_t) \leftarrow v_t(S_t) + \alpha\left(\overbrace{\underbrace{R_{t+1} + \gamma v_t(S_{t+1})}_{\text{target}} - v_t(S_t)}^{\text{TD error}}\right)$$

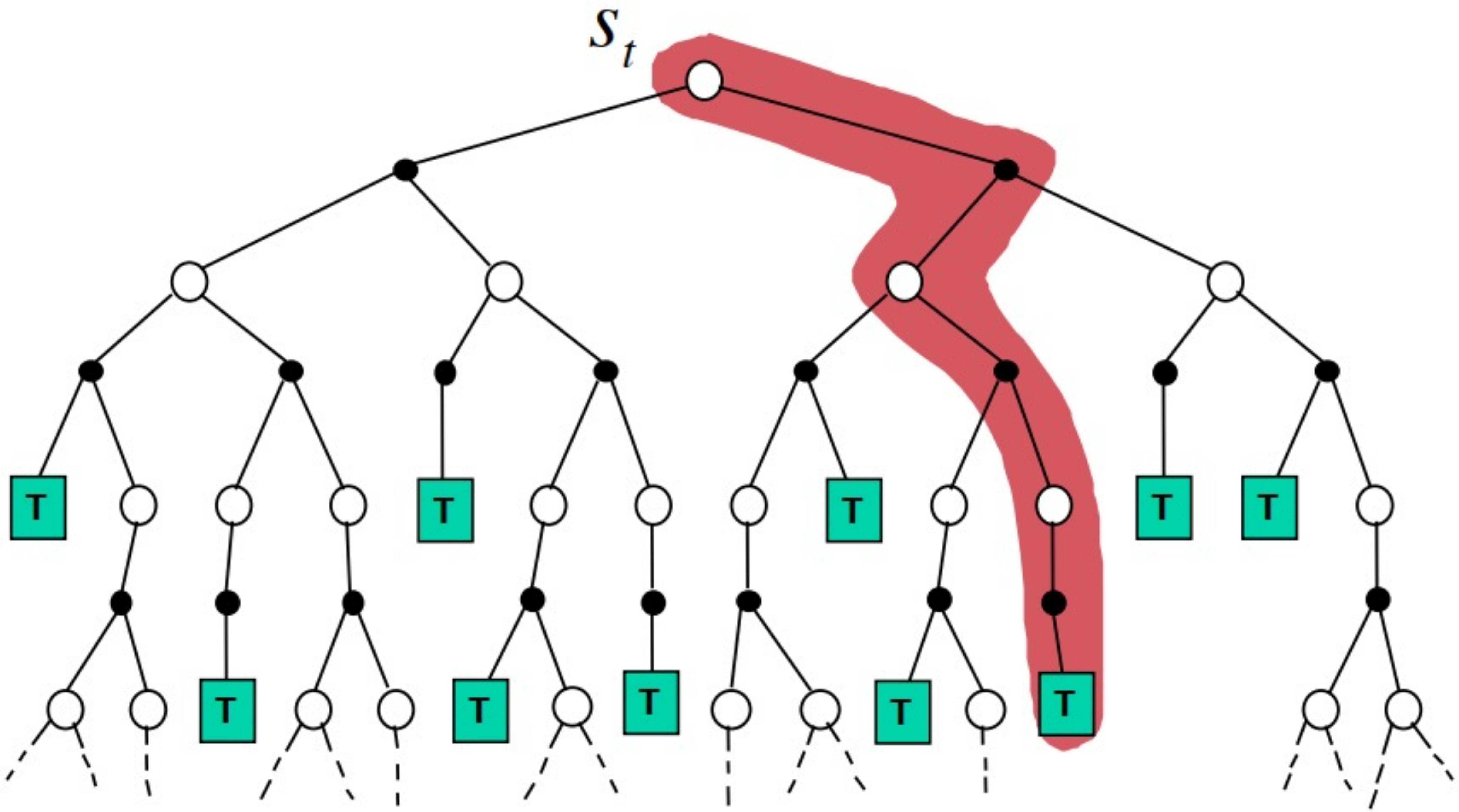
Backup (Dynamic Programming)

$$v(S_t) \leftarrow \mathbb{E}[\, R_{t+1} + \gamma v(S_{t+1}) \mid A_t \sim \pi(S_t) \,]$$



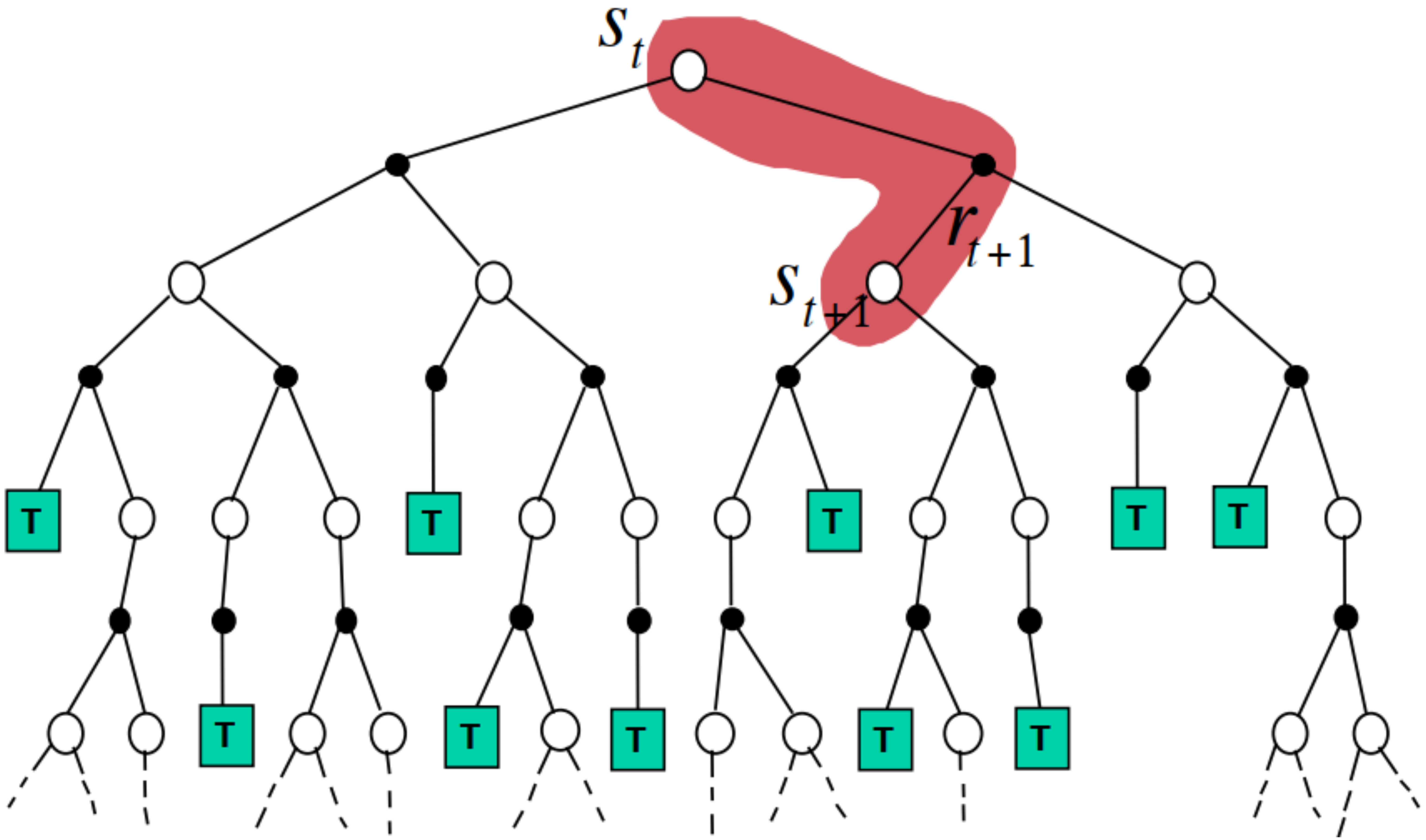
Backup (Monte Carlo)

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



Backup (Temporal Difference)

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



Bootstrapping and Sampling

- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples

TD Learning for action values

- We can apply the same idea to action values
- Temporal-difference learning for action values:

☑ Update value $q_t(S_t, A_t)$ towards estimated return $R_{t+1} + \gamma q(S_{t+1}, A_{t+1})$

$$q_{t+1}(S_t, A_t) \leftarrow q_t(S_t, A_t) + \alpha \left(\overbrace{\underbrace{R_{t+1} + \gamma q_t(S_{t+1}, A_{t+1})}_{\text{target}} - q_t(S_t, A_t)}^{\text{TD error}} \right)$$

TD vs. MC

- 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
  - MC must wait until end of episode before return is known
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments
- TD is independent of the temporal span of the prediction
  - TD can learn from single transitions
  - MC must store all predictions (or states) to update at the end of an episode
- TD needs reasonable value estimates

Temporal Difference Learning

Control

SARSA Algorithm for On-Policy Control

Algorithm:

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

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)**

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

    $S \leftarrow S'; A \leftarrow A';$

  until S **is terminal**

Off-Policy TD and Q-Learning

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

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$
- While using behavior policy $\mu(a,s)$ to generate actions
- Why is this important?
  - Learn from observing humans or other agents (e.g., from logged data)
  - Re-use experience from old policies (e.g., from your own past experience)
  - Learn about multiple policies while following one policy
  - Learn about greedy policy while following exploratory policy

Q-Learning

- Q-learning estimates the value of the greedy policy

$$q_{t+1}(s,a) = q_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t) \right)$$

- Acting greedy all the time would not explore sufficiently

Theorem

Q-learning control converges to the optimal action-value function, $q \to q^*$, as long as we take each action in each state infinitely often.

- Note: no need for greedy behavior!
- Works for any policy that eventually selects all actions sufficiently often

Q-Learning for Off-Policy Control

Algorithm:

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

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'

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

    $S \leftarrow S'$;

  until S **is terminal**