Temporal Difference Learning

Mattia Pellegrino, Ph.D Fellow
mattia.pellegrino@unipr.it

DP AND MC – RECAP

- Dynamic Programming
  - Update per step, bootstrapping
  - Need model
  - Computation cost
- Monte Carlo method
  - Update per episode
  - Model-free
  - Hard to applied to continuing task

Can we combine Dynamic Programming and Monte Carlo methods?

TEMPORAL-DIFFERENCE LEARNING

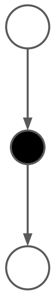
- Monte Carlo methods learn directly from episodes of experience
- Monte Carlo is classified as model-free (no knowledge of MDP transitions/rewards)
- MC learns from complete episodes
- MC use the mean return as value → value = mean return

- MC can only be applied to episodic MDPS
  - All episodes must terminate

TEMPORAL-DIFFERENCE LEARNING

- TD learning is just a few steps, not the whole trajectory
  - Different from MC method
- TD based its update in part of existing estimate (bootstrapping method)
- TD is a policy evaluation method (used to predict the value of fixed policy)

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



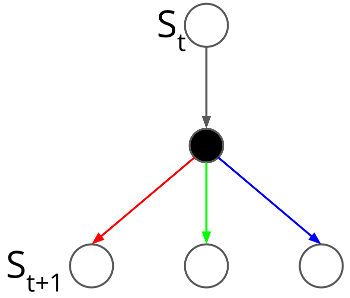
TD(0)

TEMPORAL-DIFFERENCE LEARNING

- We want to improve our estimate of V by computing these averages:

$$V_{k+1}(s) \leftarrow \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$$

- Sample 1 $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$
- Sample 2 $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$
- Sample 3 $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$



TEMPORAL-DIFFERENCE LEARNING

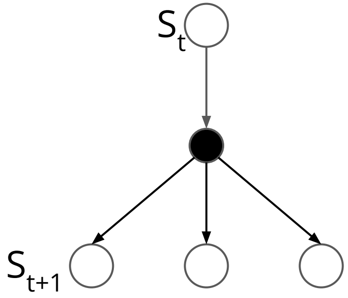
- In model-free Rl we use samples to estimate the expectation of future total rewards

- Sample 1 $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$
- Sample 2 $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

....

- Sample n $R(S_t, A_t, S_{t+1}) + \gamma V_k(S_{t+1})$

$$V_{k+1}(S_t) = \frac{1}{n} \sum_{i=0} sample_i$$



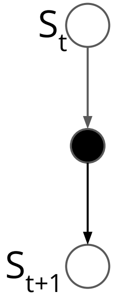
TEMPORAL-DIFFERENCE LEARNING

- TD update, one-step TD/TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha \left[\underbrace{\boxed{R_{t+1} + \gamma V(S_{t+1})}}_{\text{The target of TD method}} - V(S_t) \right]$$

- The highlighted part is a sort of error, called, *TD error*

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$



TD(0)

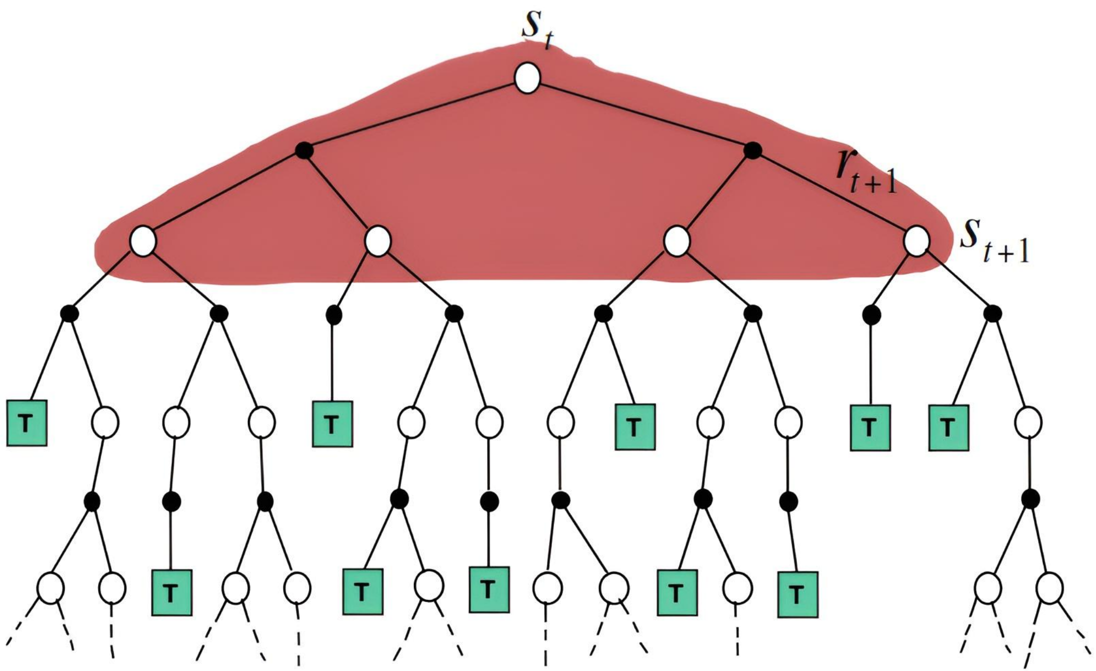
TEMPORAL-DIFFERENCE LEARNING

- Model free
- Online Learning
  - Can be applied to continuing task

- Better convergence in time
  - In practice, converge faster than Monte Carlo method

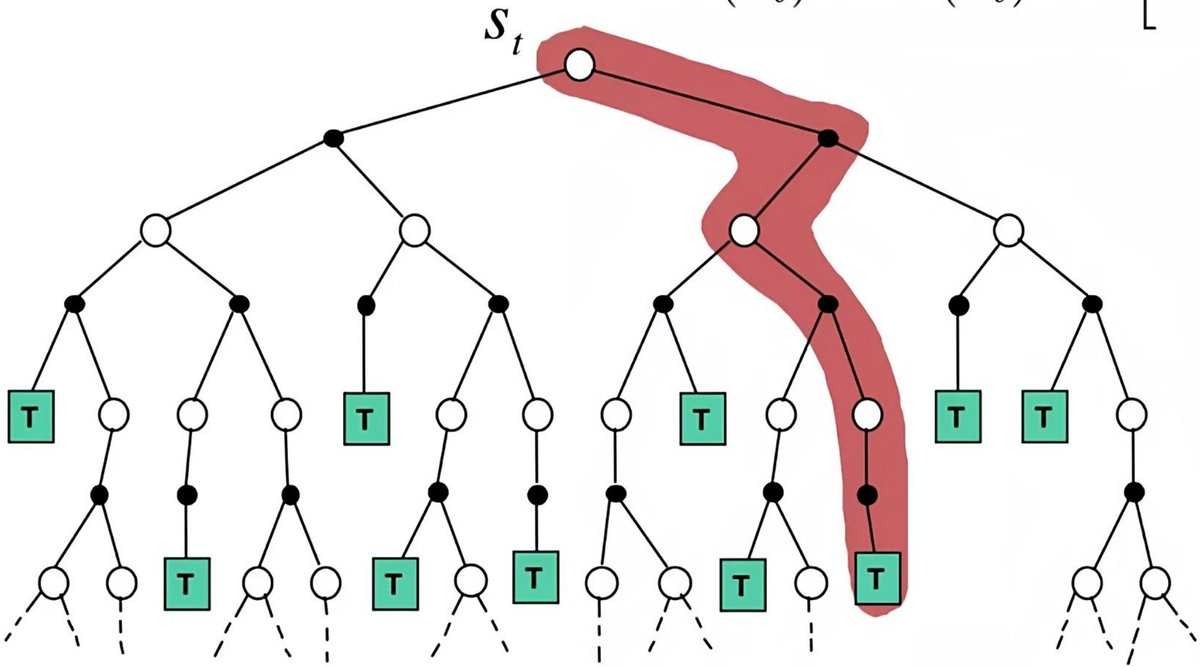
DYNAMIC PROGRAMMING

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



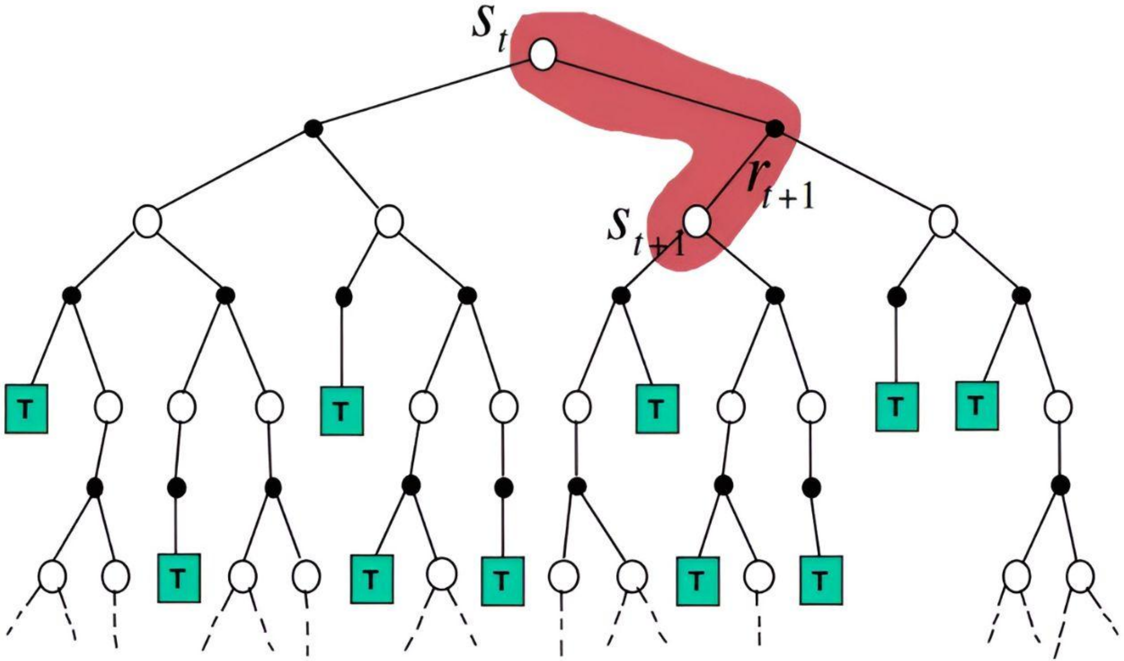
MONTE-CARLO LEARNING

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

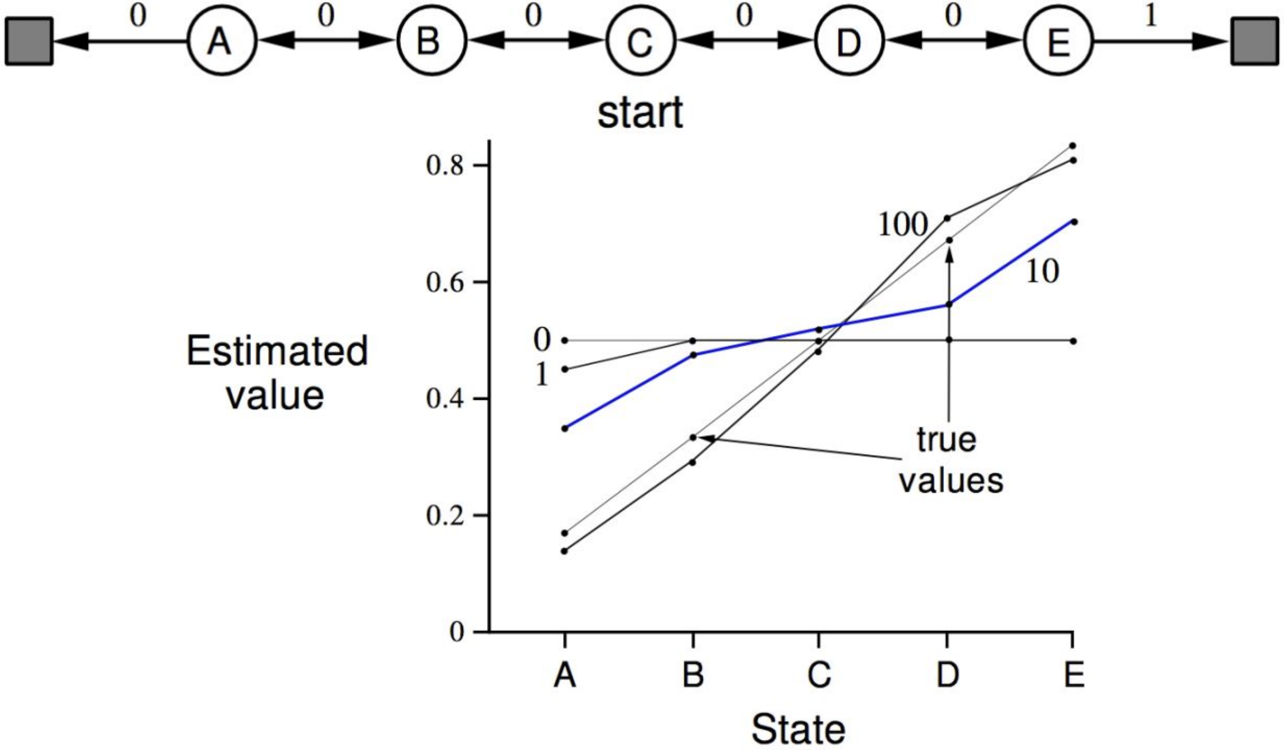


TD LEARNING

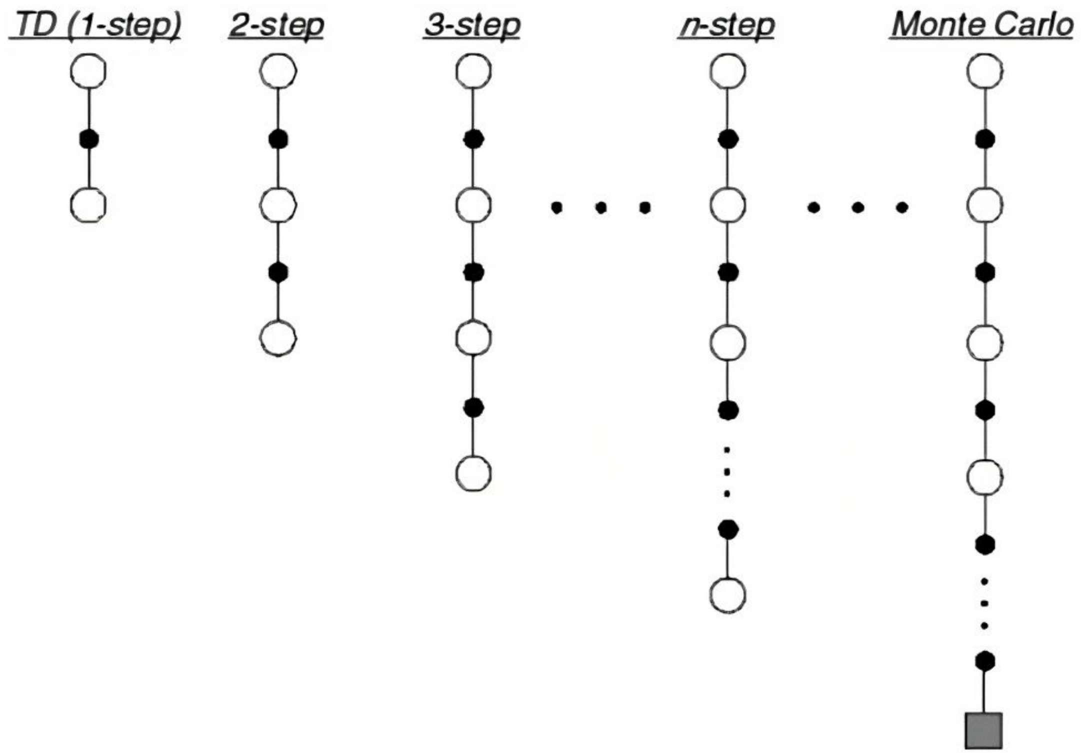
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



TD LEARNING


0
A
0
B
0
C
0
D
0
E
1
start
Estimated value
0.8
0.6
0.4
0.2
0
100
10
0
1
true values
A
B
C
D
E
State

N-STEP TD



N-STEP TD

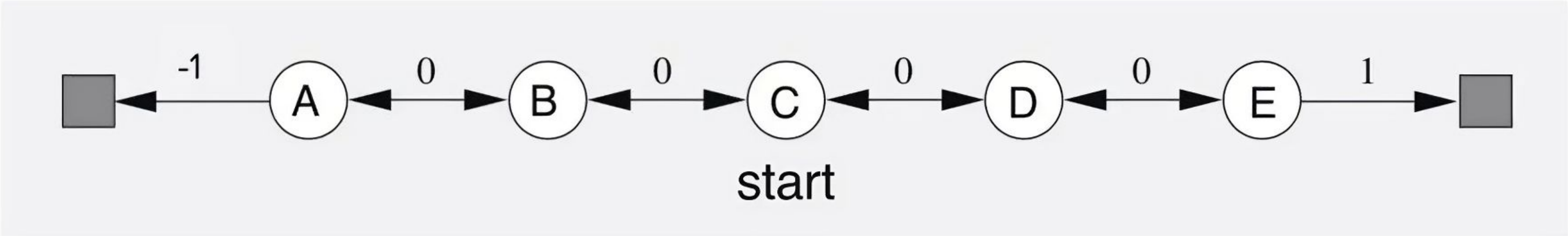
n-step TD for estimating $V \approx v_\pi$

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 $n+1$

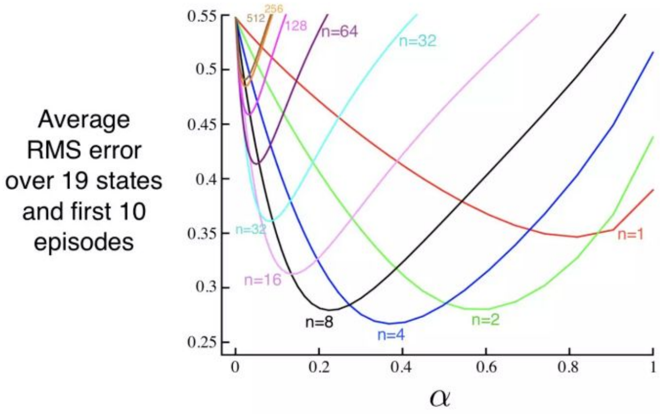
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, 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 $\tau = T - 1$

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_{t:t+n} - V_{t+n-1}(S_t) \right], 0 \leq t < T$$

N-STEP TD – RANDOM WALK



- 2 terminal states
- with 19 states instead of 5



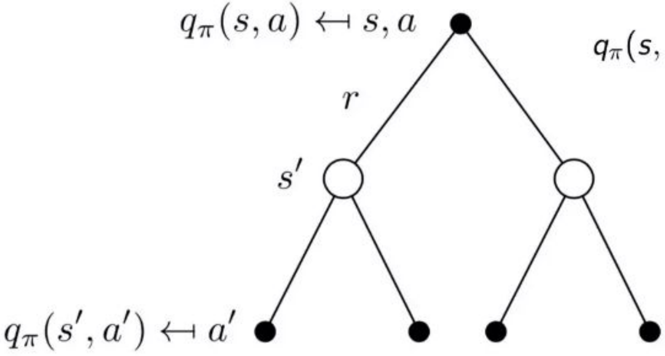
TEMPORAL-DIFFERENCE LEARNING - CONTROL

- We introduced TD learning which is used to predict the value function by one-step sample

- We can introduce two classic method in TD control:
  - Sarsa
  - Q-learning

SARSA

- Inspired by policy iteration
- Replace value function by action-value function



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

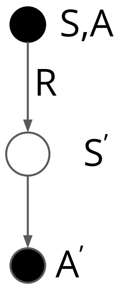
SARSA

- Inspired by policy iteration
- Replace value function by action-value function

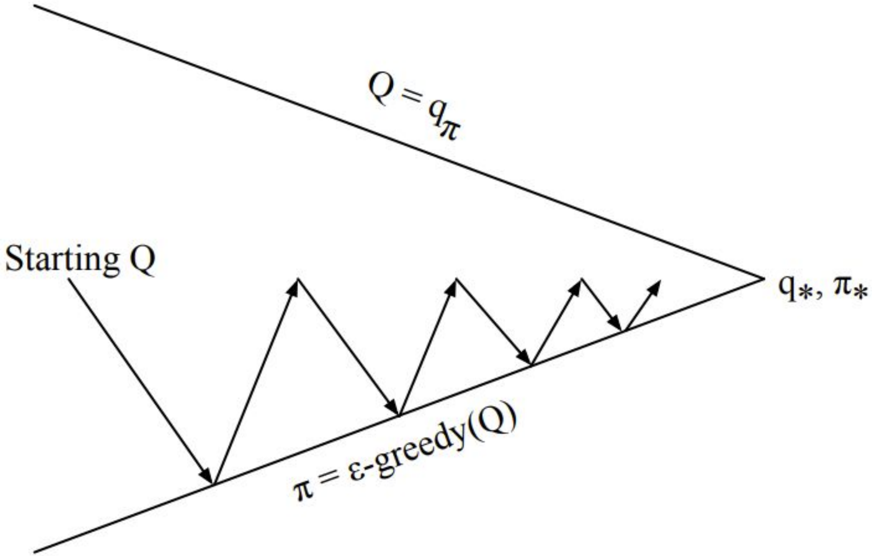
$$Q(s, a) \leftarrow Q(s, a) + \alpha\left[R(s, a, s') + \gamma Q(s', a') - Q(s, a)\right]$$

- In model-free method, we don't know the transition probability. All we need to do is to use a lot of experience sample to estimate value.

  The experience sample in Sarsa is (s,a,r,s′,a′)



SARSA



Every time-step:

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

Policy improvement ϵ-greedy policy improvement

SARSA

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\textit{terminal}, \cdot) = 0$

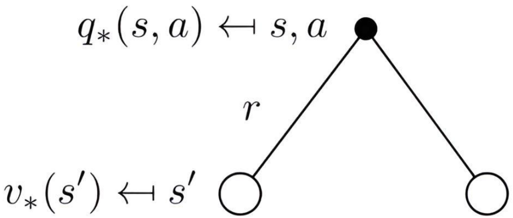
Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

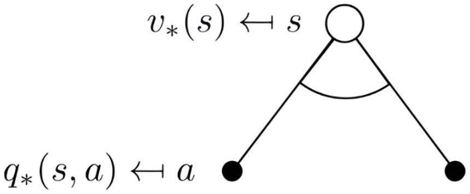
Q-Learning

- Inspired by policy iteration

$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$v_*(s) = \max_a q_*(s,a)$$





$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

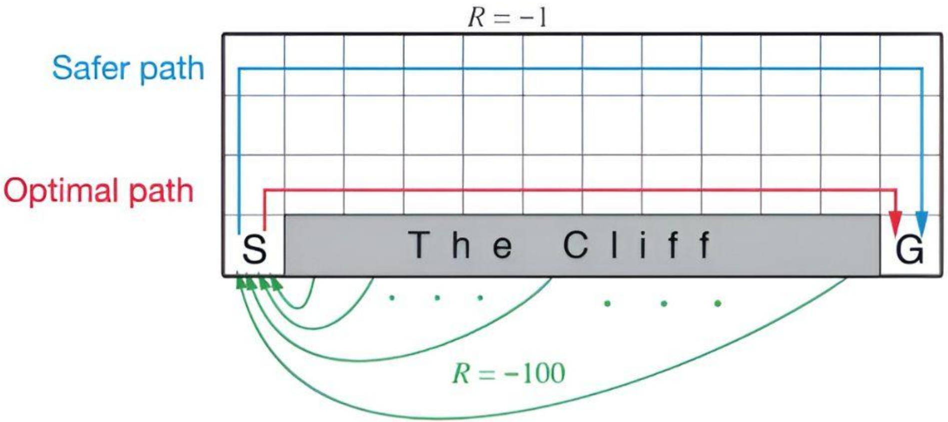
Q-Learning

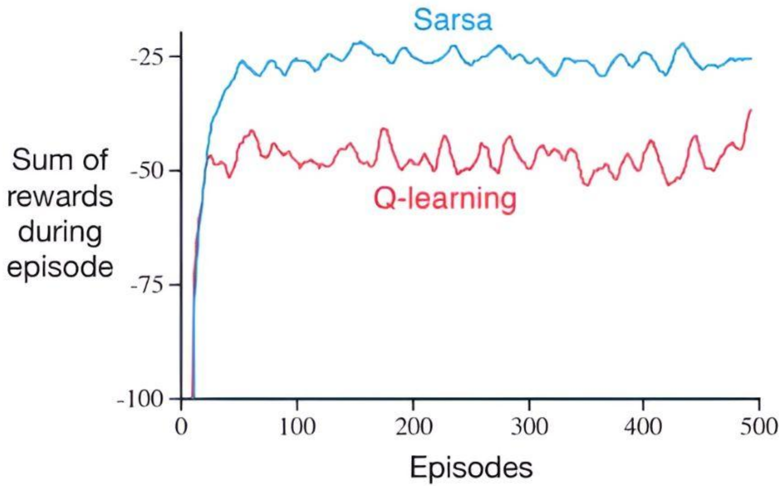
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $\boxed{Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]}$
    $S \leftarrow S'$
  until S is terminal

SARSA VS Q-LEARNING – CLIFF WALL

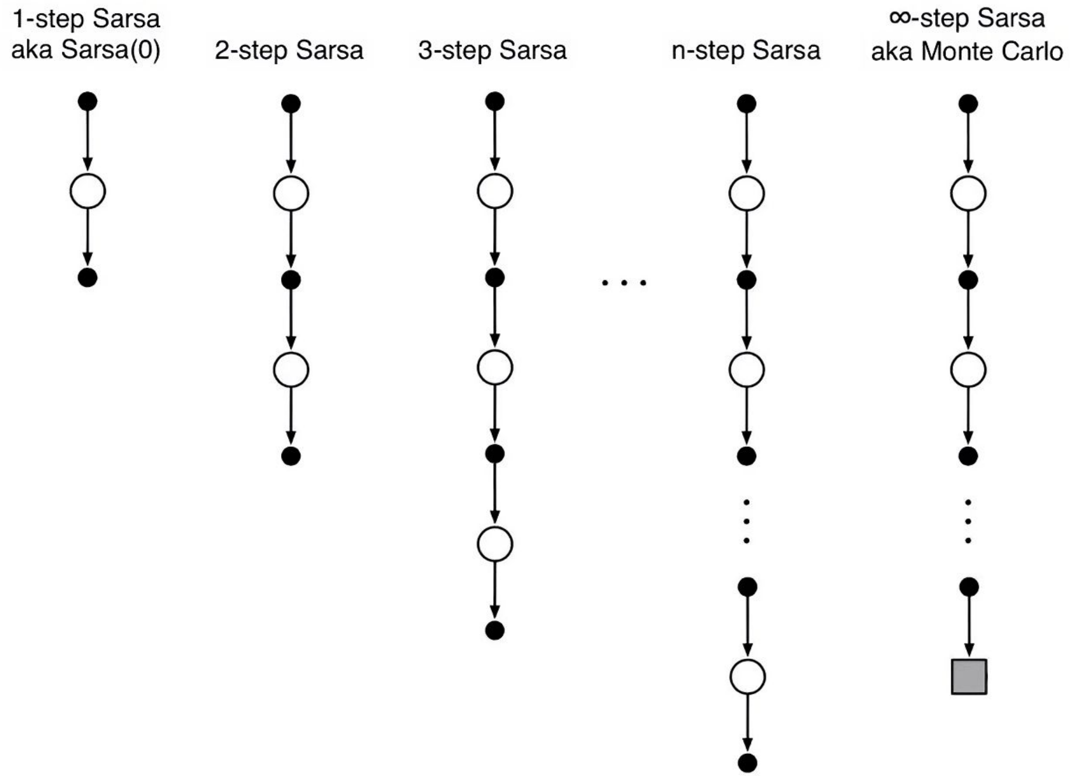




SARSA

CAN WE EVALUATE POLICY STEPS LESS THAN MONTE CARLO BUT MORE THAN ONE-STEP TD?

N-STEP SARSA



1-step Sarsa
aka Sarsa(0)
2-step Sarsa
3-step Sarsa
n-step Sarsa
∞-step Sarsa
aka Monte Carlo

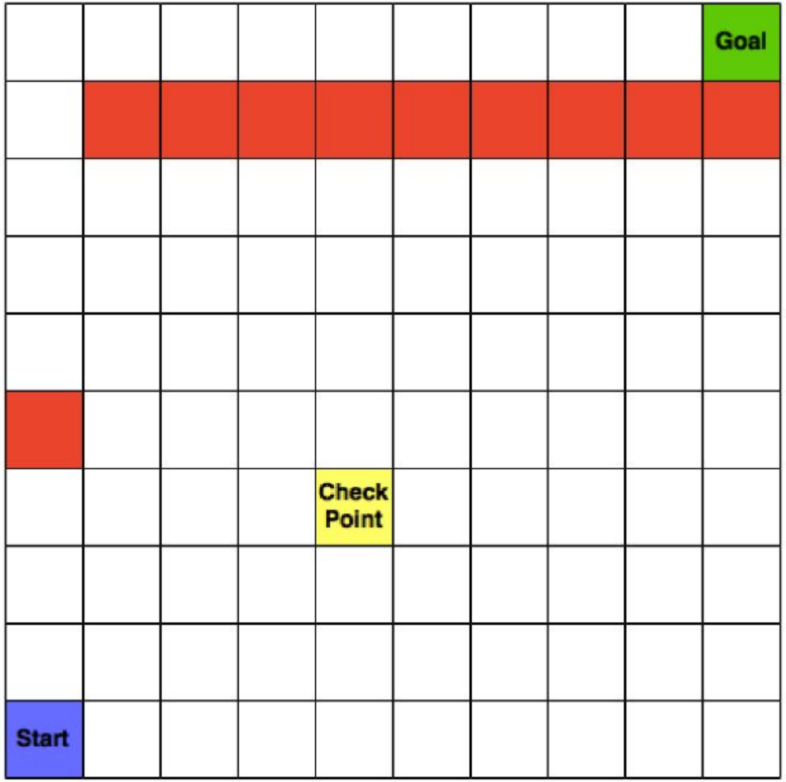
N-STEP SARSA

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

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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

EXERCISE - GRID WORLD

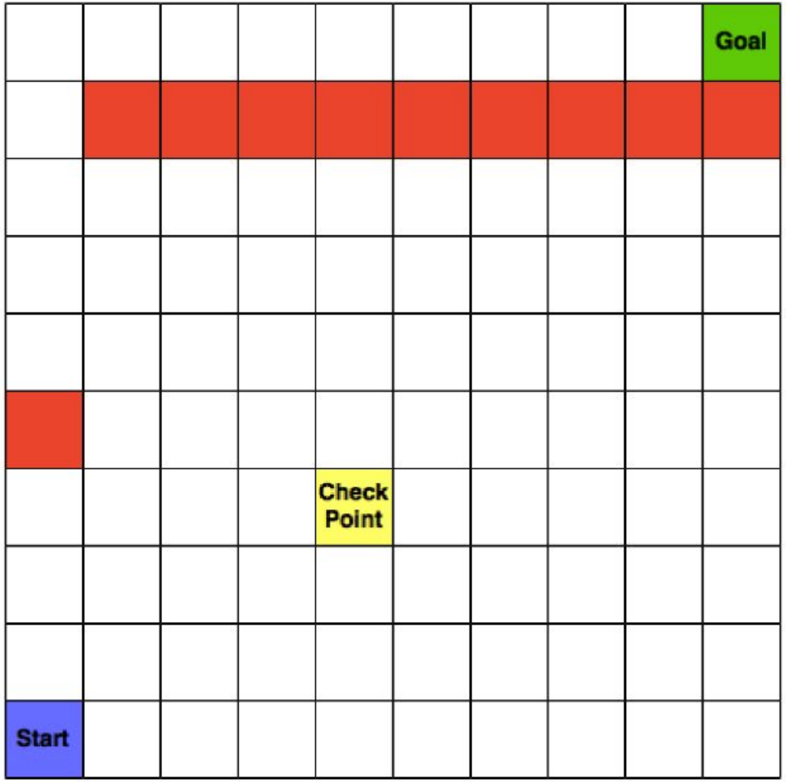


- **Rewards**
  - Step = -1
  - Wall = -10
  - Check point = 0
  - Goal = 1000
- **Action**: V in [0, V_MAX] (V=velocity)

	V - 1	V + 0	V + 1
RIGHT	0	1	2
UP	3	4	5
LEFT	6	7	8

- **Crash**:
  - Return to start
    - V = 0 & V_MAX = 3
- **CheckPoint**: V_MAX = 5

EXERCISE - GRID WORLD



- Analyze the source code (available at: https://github.com/sowide/reinforcement_learning_course [23-05_exercise folder])and add the required comments to explain its behavior
- Create a graph that shows the trend of the reward (y-axis) as the alpha (x-axis) and the number of steps vary. Keep epsilon fixed at 0.1. The number of episodes is chosen at will.
- Deliver the modified source code: https://shorturl.at/JRUX5 [name_surname.zip]