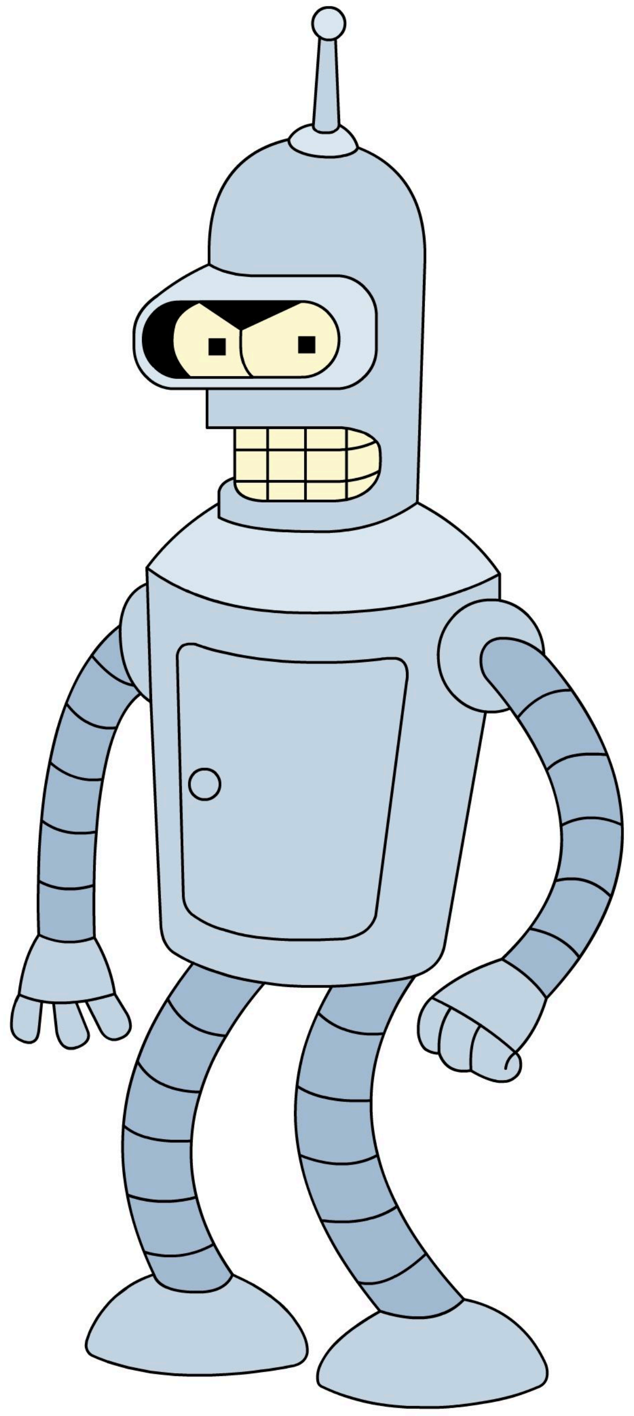
Reinforcement Learning

HSE, autumn - winter 2022

Lecture 3: Model-free RL



Sergei Laktionov
slaktionov@hse.ru
LinkedIn

Background

1. Sutton & Barto, Chapter 5 + 6 + 7
2. RL for Finance Book, Chapter 11 + 12
3. Practical RL course by YSDA, week 3
4. DeepMind course, lectures 5 + 6

Recap: Objective

Let T is a final time step. If $T < \infty$ then environment is called *episodic*.

Cumulative reward is called a return or reward-to-go. Note that in general it is a random variable.

Episode length

$$\boxed{G_t} = \boxed{R_t} + \boxed{\gamma} R_{t+1} + \gamma^2 R_{t+2} + \ldots = \sum_{s=t}^{\boxed{T}} \gamma^{s-t} R_s$$

Immediate reward

Discount factor

$$\pi^* = argmax_\pi J(\pi) = argmax_\pi \mathbb{E}_\pi[G_0]$$

Recap: Bellman Equations

Bellman expectation equations:

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{r,s'} p(r, s' \mid s, a)\left[r + \gamma V_\pi(s')\right]$$

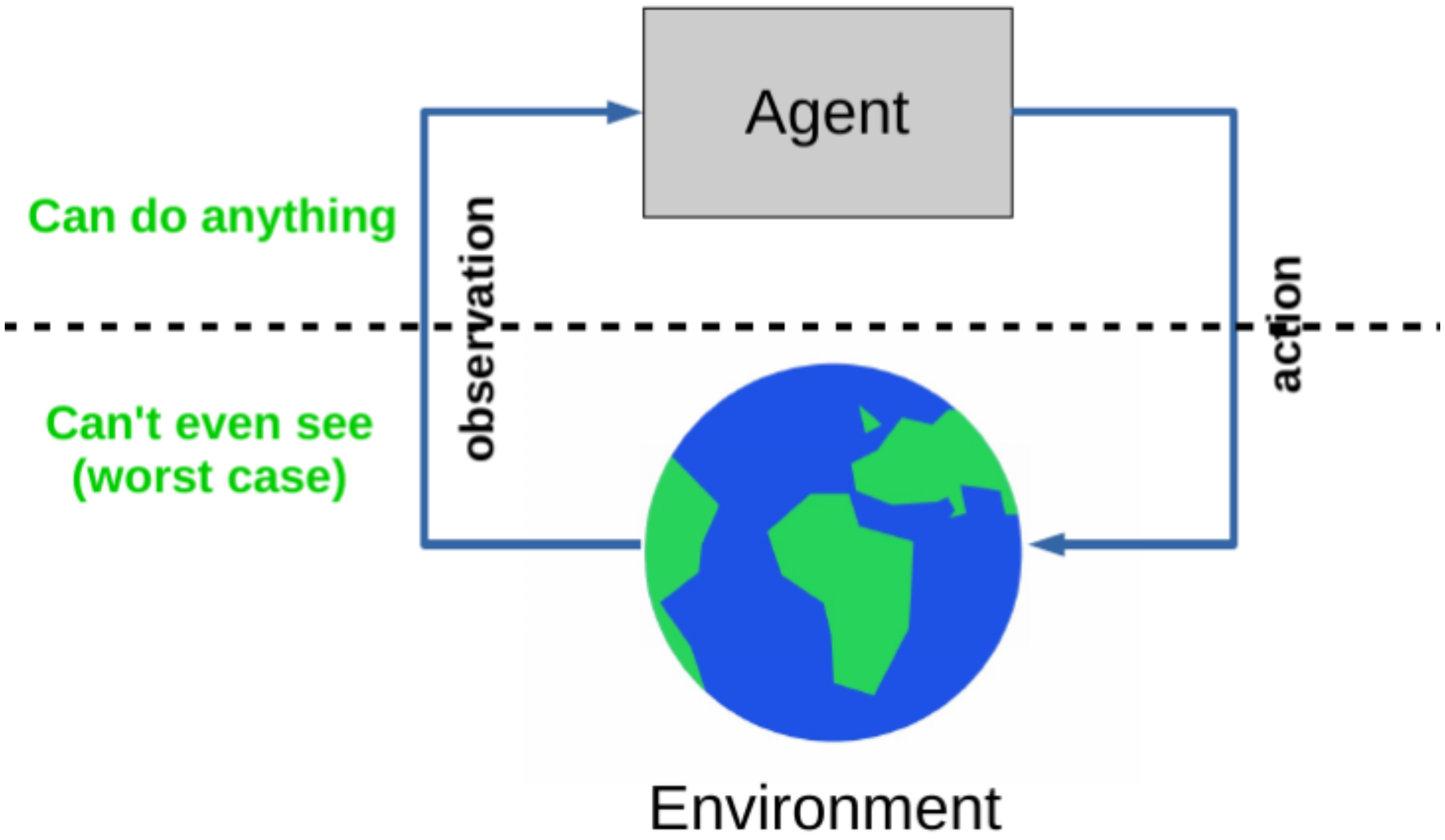
$$Q_\pi(s, a) = \sum_{r,s'} p(r, s' \mid s, a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') Q_\pi(s', a')\right]$$

Bellman optimality equations:

$$V^*(s) = V_{\pi^*}(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V^*(s')]$$

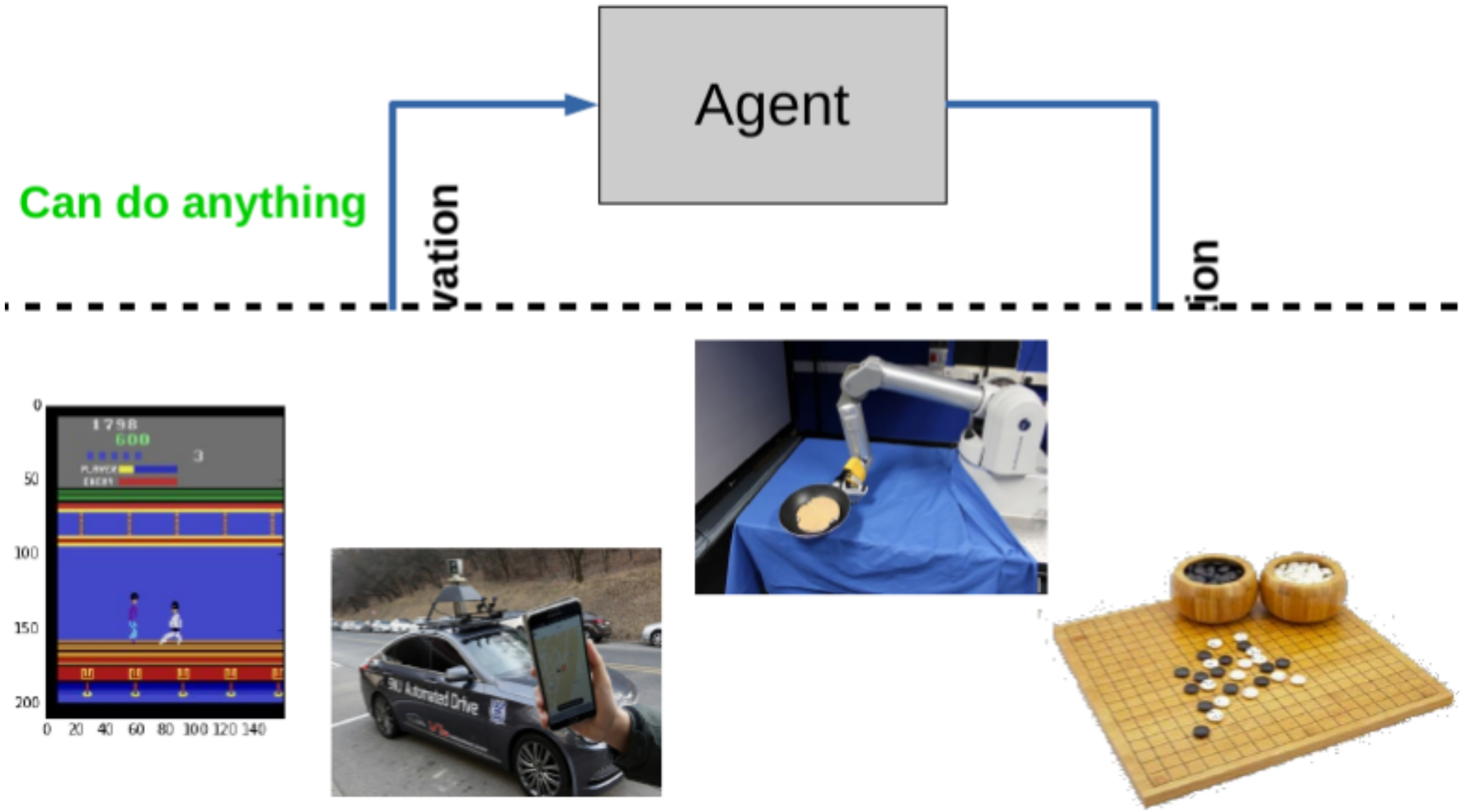
$$Q^*(s, a) = Q_{\pi^*}(s, a) = \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} Q^*(s', a')\right]$$

Decision Processes



Source

Decision Processes



Source

Dynamic Programming Algorithms

$$V_{k+1}(s) = \mathbb{E}\left[R_t + \gamma V_k(S_{t+1}) \,|\, S_t = s, A_t \sim \pi(\,.\,|\,S_t)\right]$$ (policy evaluation)

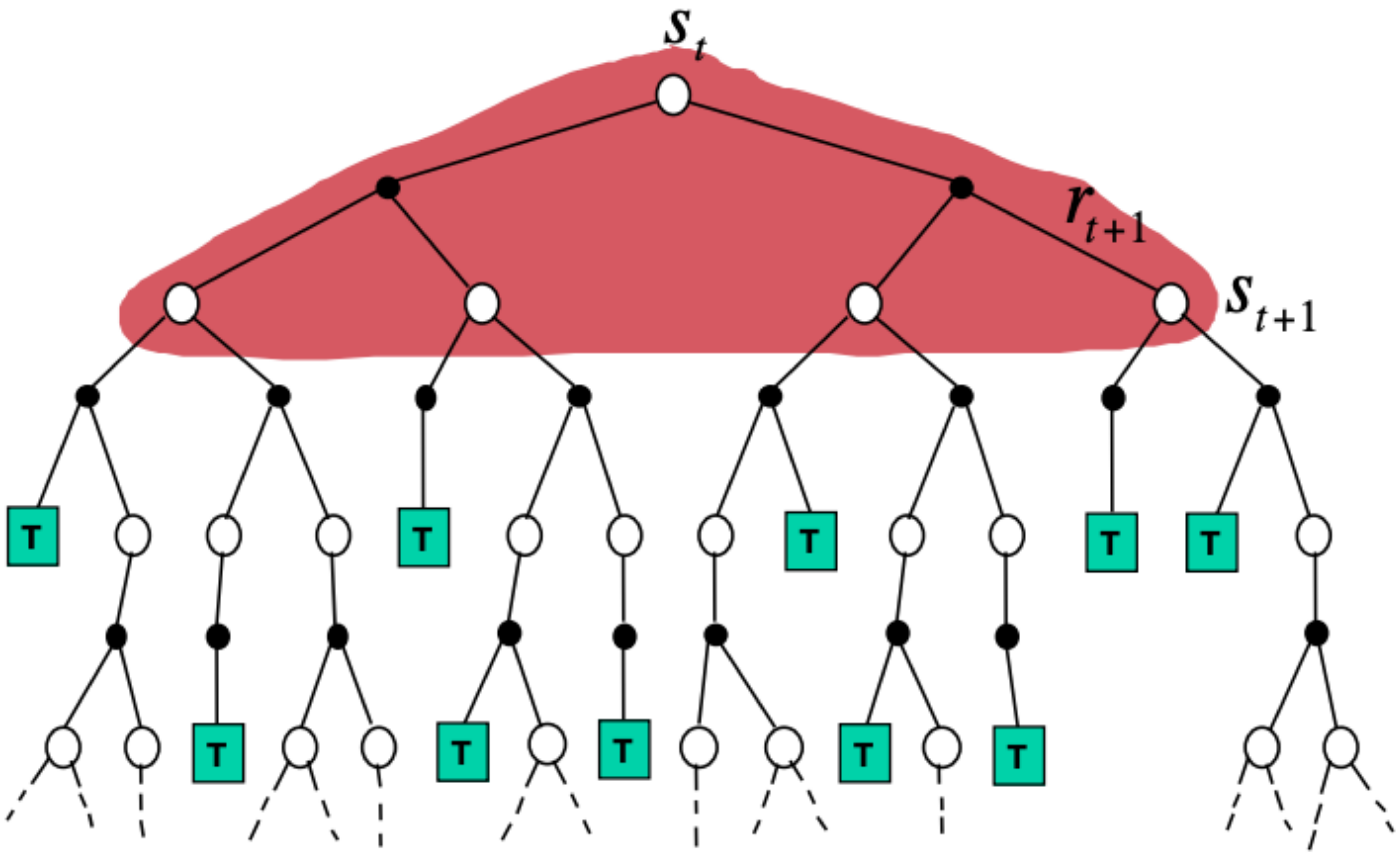
$$V_{k+1}(s) = \max_a \mathbb{E}\left[R_t + \gamma V_k(S_{t+1}) \,|\, S_t = s, A_t = a\right]$$ (value iteration)

$$Q_{k+1}(s,a) = \mathbb{E}\left[R_t + \gamma Q_k(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a\right]$$ (policy evaluation)

$$Q_{k+1}(s,a) = \mathbb{E}\left[R_t + \gamma \max_a Q_k(S_{t+1}, a) \,|\, S_t = s, A_t = a\right]$$ (value iteration)

Dynamic Programming Backup

$$V_{k+1}(s) = \mathbb{E}\left[R_t + \gamma V_k(S_{t+1}) \,|\, S_t = s, A_t \sim \pi(\,.\,|\,S_t)\right]$$



Source

But what should we do if the environment's dynamic is no more available?

Monte-Carlo Policy Evaluation

Let us estimate $\mathbb{E}\left[R_t + \gamma G_{t+1} \,|\, S_t = s, A_t \sim \pi(\,.\,|\,S_t)\right]$ as the sample mean of the returns:

Monte-Carlo Policy Evaluation

Let us estimate $\mathbb{E}\left[R_t + \gamma G_{t+1} \,|\, S_t = s, A_t \sim \pi(\,.\,|\,S_t)\right]$ as the sample mean of the returns:

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:
- $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
- $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
- Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$
- Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
    - Append G to $Returns(S_t)$
    - $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Source

Monte-Carlo Policy Evaluation

Let us estimate $\mathbb{E}\left[R_t + \gamma G_{t+1} \,|\, S_t = s, A_t \sim \pi(\,.\,|S_t)\right]$ as the sample mean of the returns:

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

      Append G to $Returns(S_t)$

      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Source

Same idea is straightforwardly applicable for estimation $Q_\pi(s,a) = \mathbb{E}\left[R_t + \gamma G_{t+1} \,|\, S_t = s, A_t = a\right]$ but we need that each state-action pair is visited an infinite number of times in the limit of an infinite number of episodes.

Exploration vs Exploitation

The only general way to ensure that all actions are selected infinitely often is for the agent to continue to select them. Firstly we consider on-policy methods which attempt to evaluate or improve the policy that us used to made decisions. In on-policy control methods the policy is generally soft:

$$\pi(a \mid s) > 0 \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

Exploration vs Exploitation

The only general way to ensure that all actions are selected infinitely often is for the agent to continue to select them. Firstly we consider on-policy methods which attempt to evaluate or improve the policy that us used to made decisions. In on-policy control methods the policy is generally soft:

$$\pi(a \mid s) > 0 \, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

Consider so-called ε-greedy policy:

$$\pi = \begin{cases} \text{select random action with probabily } \varepsilon \\ \text{select greedy action with probabily } 1 - \varepsilon \end{cases}$$

Among ε-soft policies, ε-greedy policies are in some sense those that are closest to greedy.

Model-free Control

$p(r, s' | s, a)$ is not available anymore or extremely hard to obtain.

How to recover optimal policy in this case:

1. If Q^* is known: $\pi(s) = argmax_a Q^*(s, a)$
2. If V^* is known: $\pi(s) = argmax_a \sum_{r,s'} \cancel{p(r, s' | s, a)}[r + \gamma V^*(s')]$

No more available

Model-free Control

$p(r, s' | s, a)$ is not available anymore or extremely hard to obtain.

How to recover optimal policy in this case:

1. If $Q*$ is known: $\pi(s) = argmax_a Q*(s, a)$
2. If $V*$ is known: $\pi(s) = argmax_a \sum_{r,s'} \cancel{p(r, s' | s, a)}[r + \gamma V*(s')]$

No more available

$V*$ is useless for control problem in a model-free setting

Monte-Carlo Control

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

  $\pi \leftarrow$ an arbitrary ε-soft policy

  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

      Append G to $Returns(S_t, A_t)$

      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Source

Monte-Carlo Methods

Disadvantages:

1. We have to wait until an episode ends before we can learn
2. Return can have high variance

Temporal-Difference Learning

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas:

1. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.

2. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).

Temporal-Difference Learning

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas:

1. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.

2. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).

General idea: replace the expectation E with moving average update for $\alpha \in (0,1]$

TD error

$$E \leftarrow \alpha \hat{E} + (1-\alpha)E = E + \alpha\boxed{(\hat{E} - E)},$$

Target

where $\hat{E}$ is better than E in some sense.

TD Prediction

1. MC: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] = \alpha G_t + (1-\alpha)V(S_t)$
2. TD(0): $V(S_t) \leftarrow V(S_t) + \alpha[R_t + \gamma V(S_{t+1}) - V(S_t)] = \alpha[R_t + \gamma V(S_{t+1})] + (1-\alpha)V(S_t)$

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

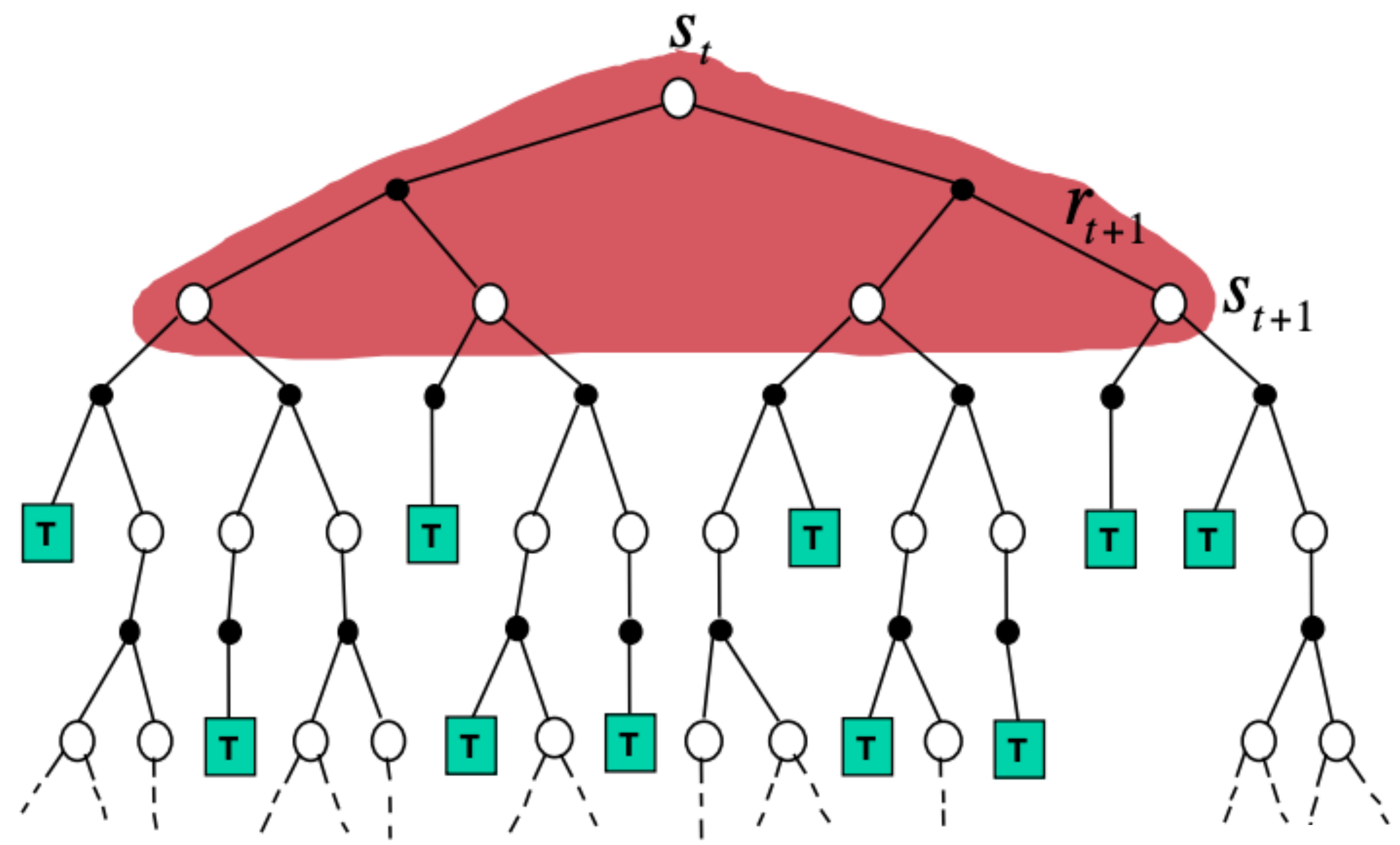
TD(0)

Source

Backup Diagrams

DP

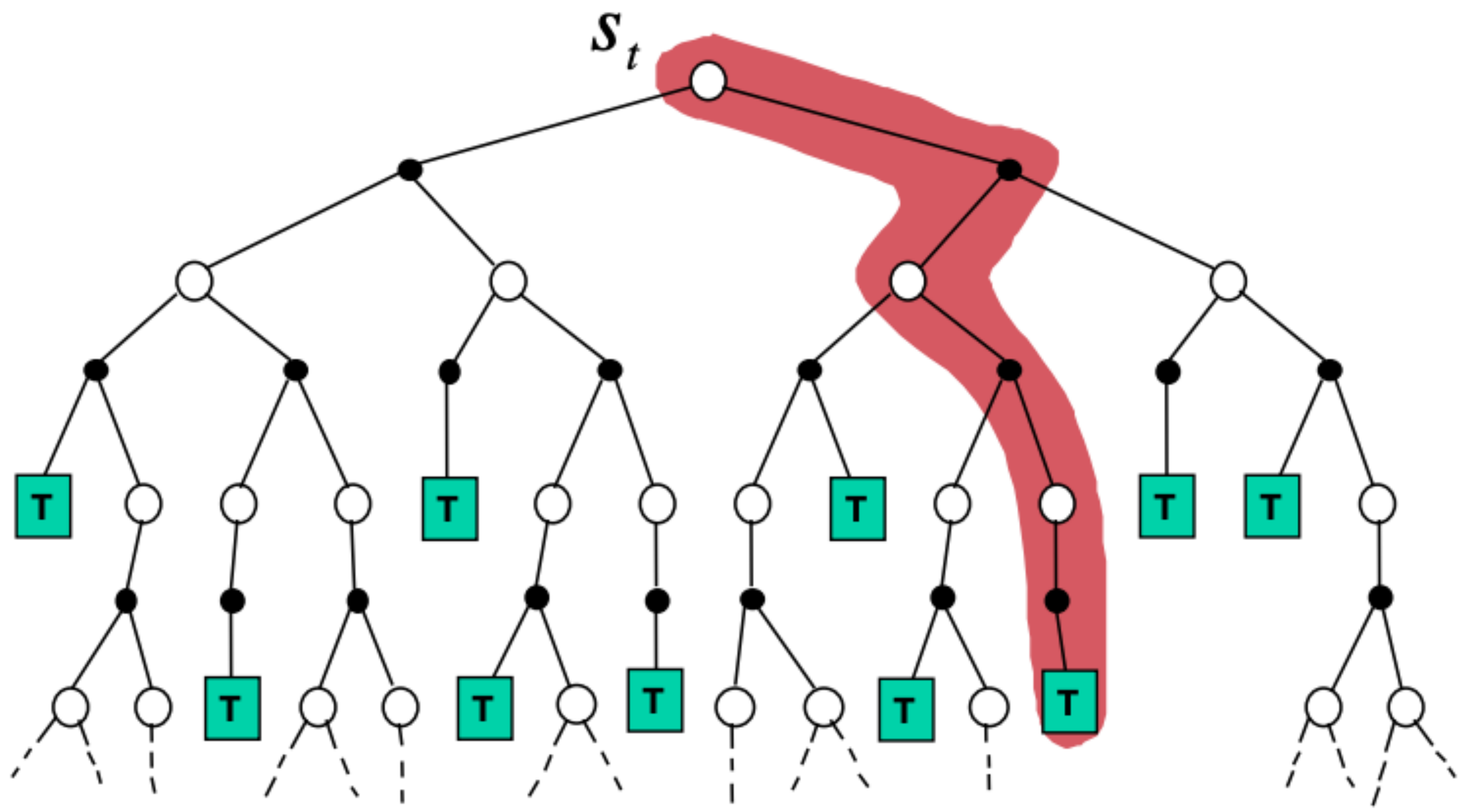
$$v(S_t) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid A_t \sim \pi(S_t)\right]$$



Source

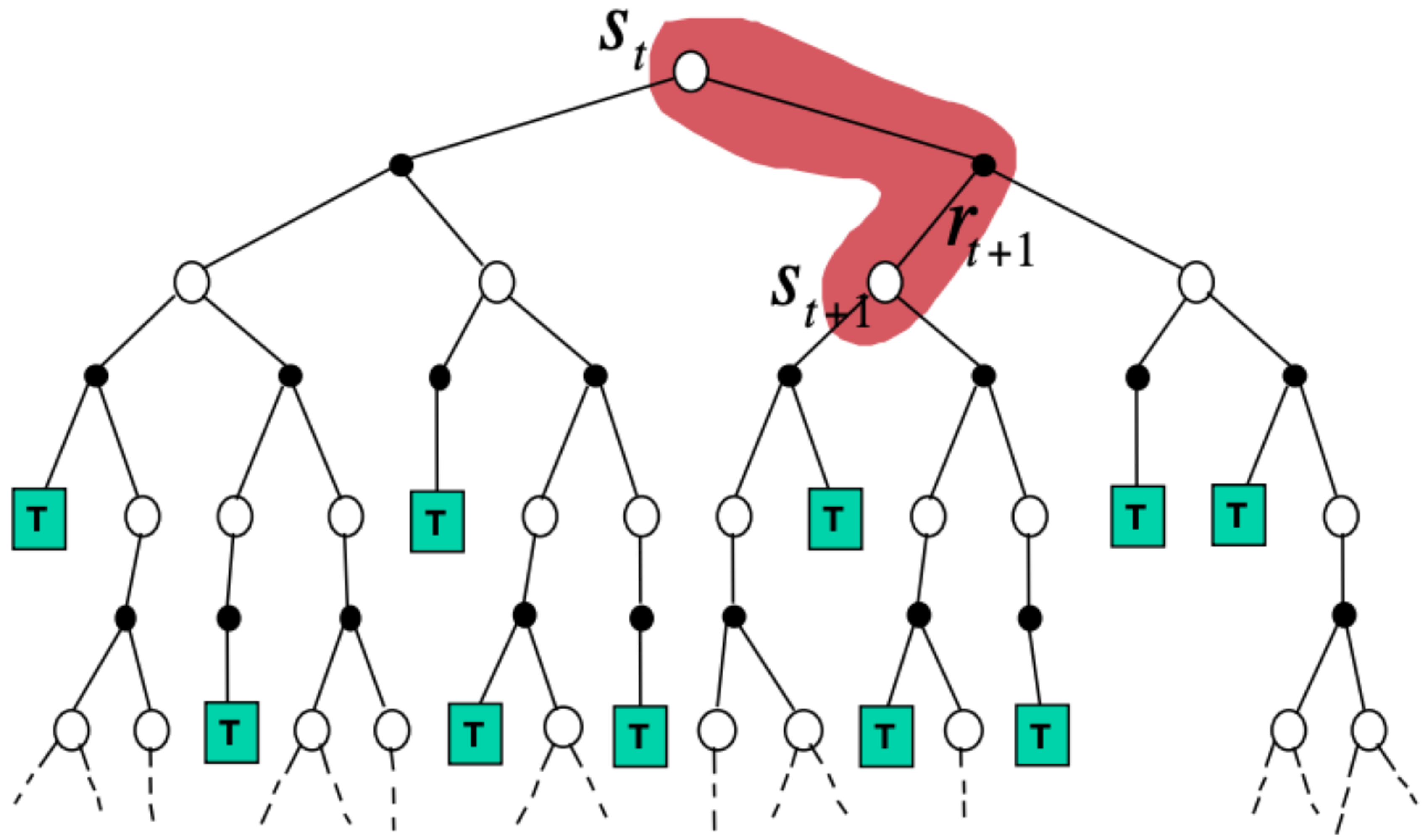
MC

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



TD(0)

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



SARSA: On-policy TD Control

We turn now to the use of TD prediction methods for the control problem. As usual, we follow the pattern of generalized policy iteration (GPI), only this time using TD methods for the evaluation or prediction part.

We can apply TD to Q:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

This algorithm is known as SARSA, because it uses $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$

SARSA: On-policy TD Control

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

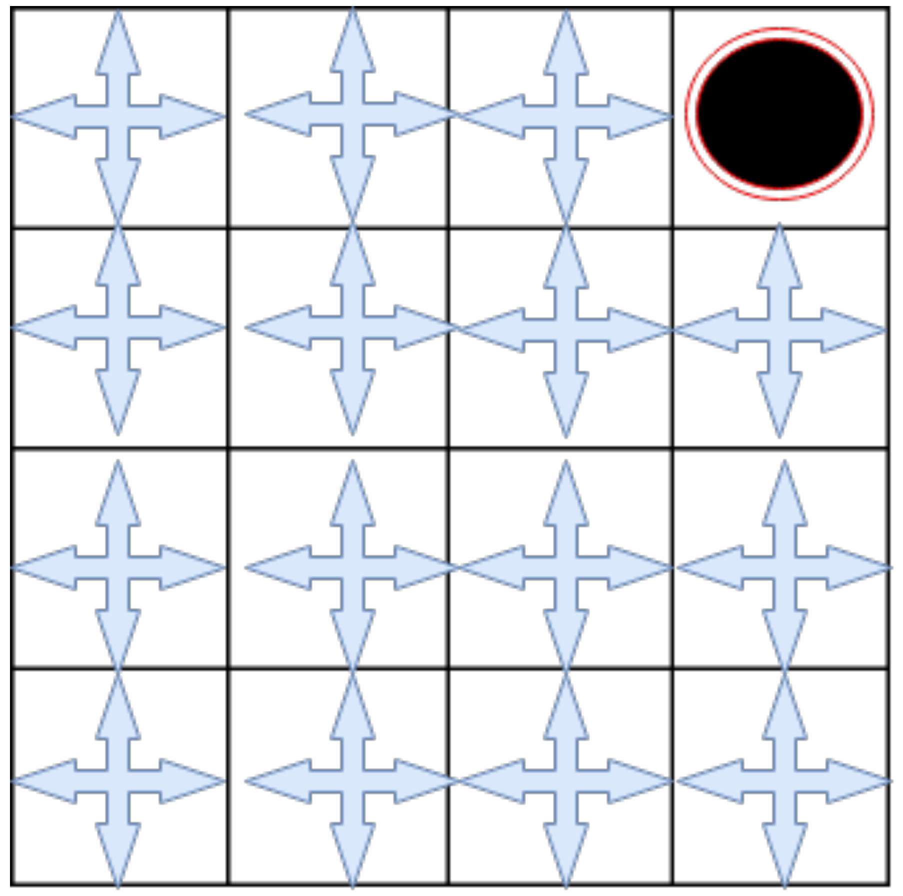
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

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

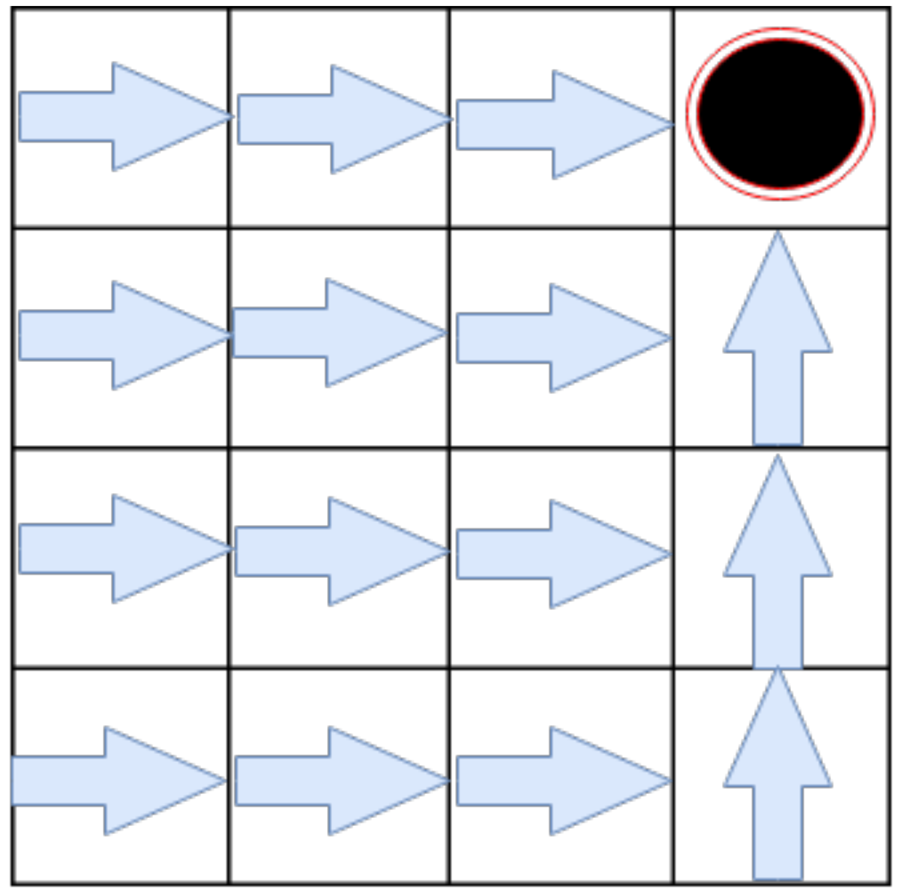
Source

Q-learning: Off-policy TD Control

Distinguishing feature of on-policy methods is that they estimate the value of a policy while using it for control. In off-policy methods these two functions are separated. The policy used to generate behaviour, called the behaviour policy, may in fact be unrelated to the policy that is evaluated and improved, called the target policy. An advantage of this separation is that the target policy may be deterministic (e.g., greedy), while the behaviour policy can continue to sample all possible actions.



Behavior Policy



Target Policy

Source

Q-learning: Off-policy TD Control

Recall the Bellman optimality equation for Q^* and apply it as an update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

In this case, the learned action-value function, Q, directly approximates Q^*, the optimal action-value function, independent of the policy being followed.

Q-learning: Off-policy TD Control

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

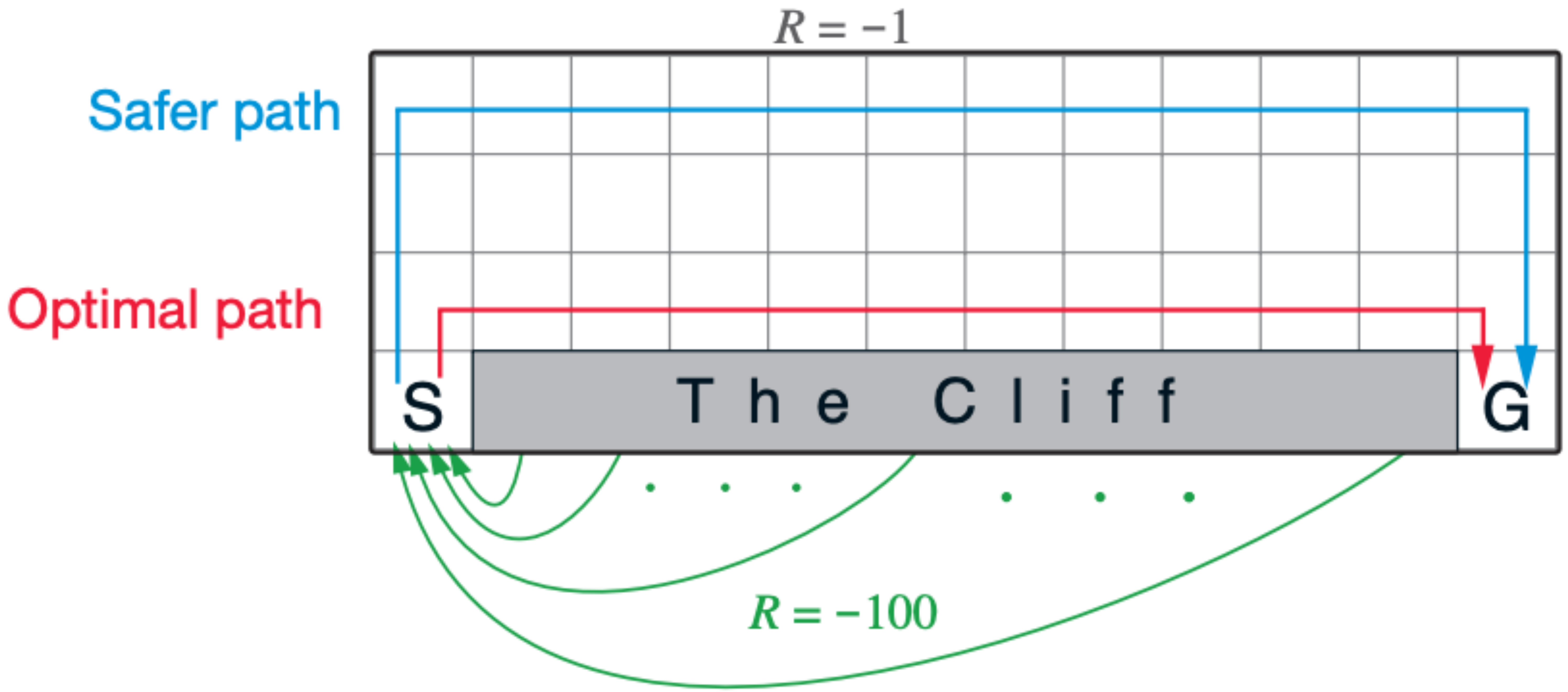
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\textit{terminal}, \cdot) = 0$

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

Source

Example: Cliff World

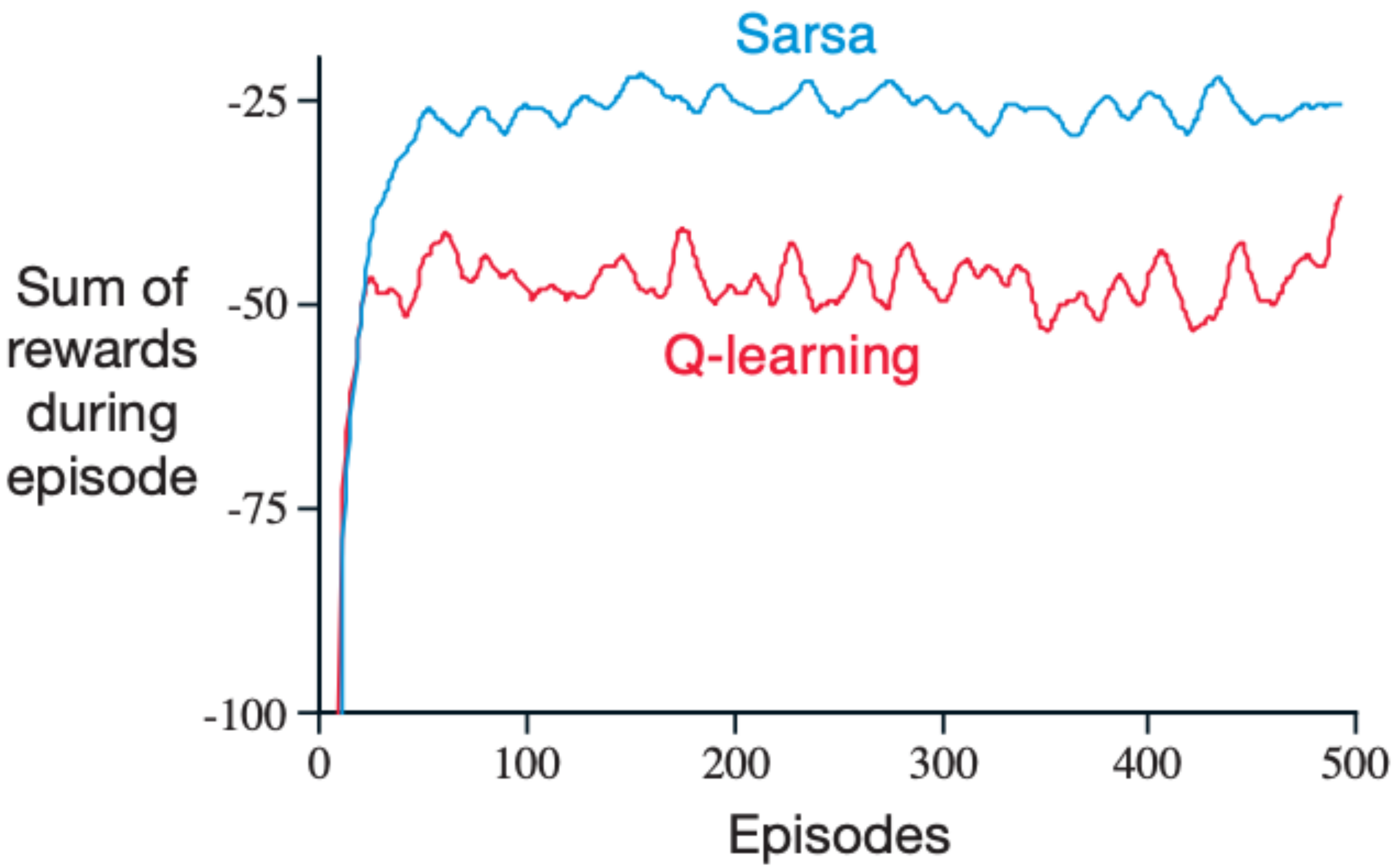
$\gamma = 1, \varepsilon = 0.1$. Agent gets -1 for each step.



Source

Which trajectory is learned by Q-learning?

Example: Cliff World



Source

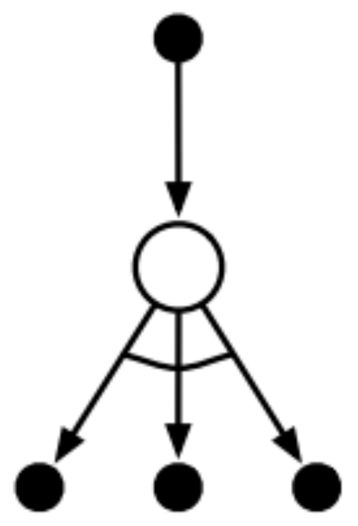
Of course, if ε were gradually reduced, then both methods would asymptotically converge to the optimal policy.

Expected SARSA

Consider the learning algorithm that is just like Q-learning except that instead of the maximum over next state–action pairs it uses the expected value, taking into account how likely each action is under the current policy.

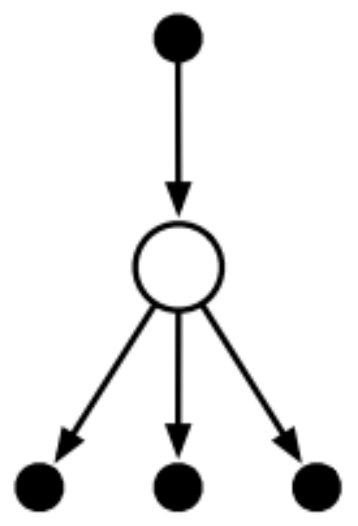
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \mathbb{E}_\pi Q(S_{t+1}, a) - Q(S_t, A_t)] =$$

$$= Q(S_t, A_t) + \alpha[R_t + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)]$$



Q-learning

Source



Expected Sarsa

Source

Maximization Bias

The overestimation of the Q-function by the algorithms which can be caused by several reasons:

1. Maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to a significant positive bias.

2. Due to using the same samples (plays) both to determine the maximizing action and to estimate its value.

Maximization Bias and Double Q-learning

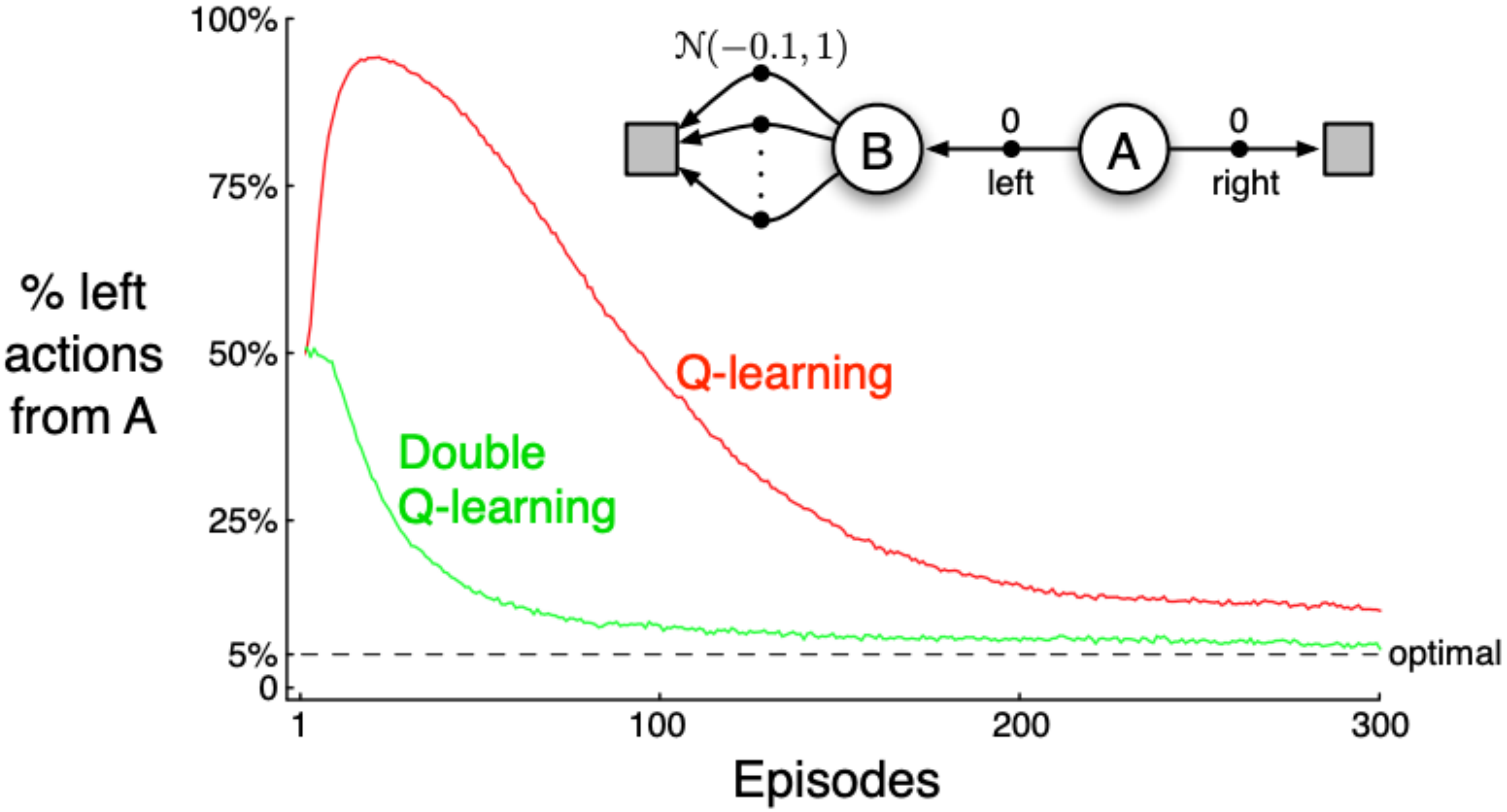
Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
    else:
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Source

Q-Learning vs Double Q-learning



Source

On-policy vs Off-Policy

On-policy learning:

- Learn about behaviour policy π from experience sampled from π

Off-policy learning:

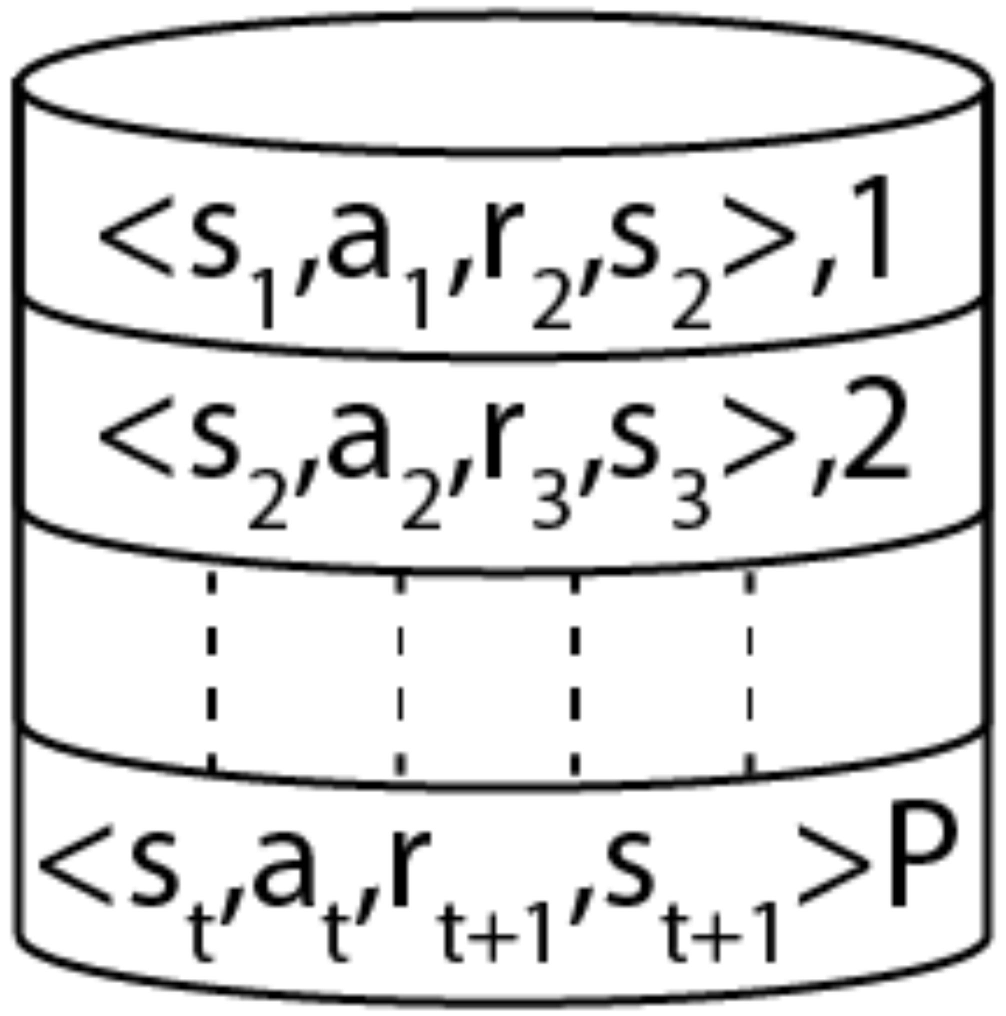
- Learn about target policy π from experience sampled from μ
- Learn ‘counterfactually’ about other things you could do: “what if...?”

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $V_\pi(s)$ or $Q_\pi(s,a)$ while using behaviour policy $\mu(a|s)$ to generate actions
- Why?
  - Learn from observing humans or other agents (e.g., from logged data)
  - Reuse 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

Experience Replay Buffer

- On each step store $< s, a, r, s' >$ in the buffer
- Sample n random transitions from the buffer
- Train on them



Source

Advantages:

- No need to revisit same states many times
- Make the estimators consistent with the current policy, update estimators
- Decorrelate update samples to maintain i.i.d. assumption

Disadvantages:

- Not applicable for the on-policy learning

N-step Bootstrapping

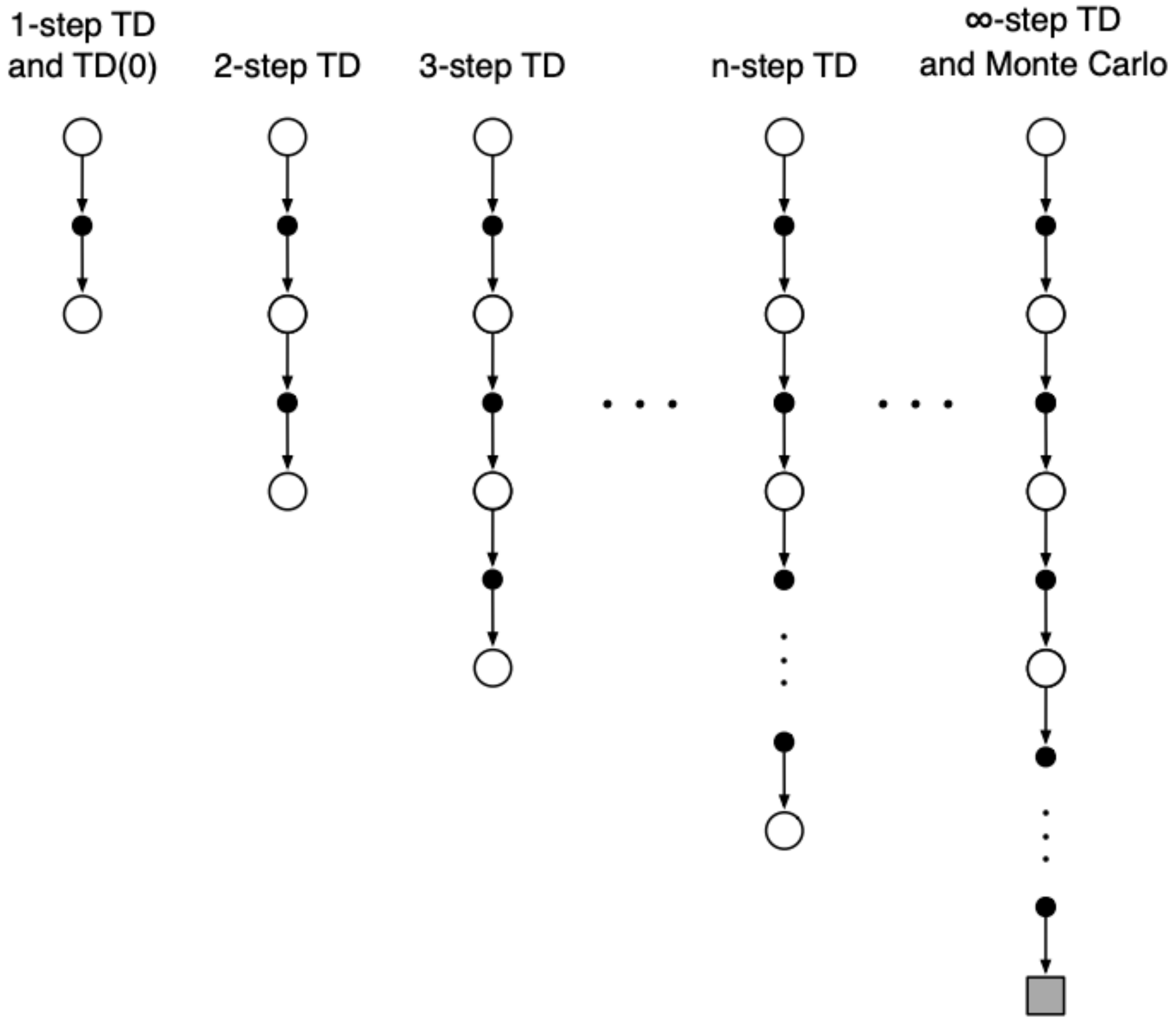
- TD uses value estimates which might be inaccurate
- In addition, information can propagate back quite slowly
- In MC information propagates faster, but the updates are noisier
- We can go in between TD and MC

N-step Prediction

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 $\tau = T - 1$



Source

N-step Control

General formula:

$$Q(S_t, A_t) \leftarrow \alpha \hat{Q}(S_t, A_t) + (1 - \alpha)Q(S_t, A_t)$$

N-step SARSA:

$$\hat{Q}(S_t, A_t) = \sum_{\tau=t}^{t+n-1} \gamma^{\tau-t} R_t + \gamma^n Q(S_{t+n}, A_{t+n})$$

N-step Q-Learning:

$$\hat{Q}(S_t, A_t) = \sum_{\tau=t}^{t+n-1} \gamma^{\tau-t} R_t + \gamma^n \max_a Q(S_{t+n}, a)$$

Thank you for your attention!