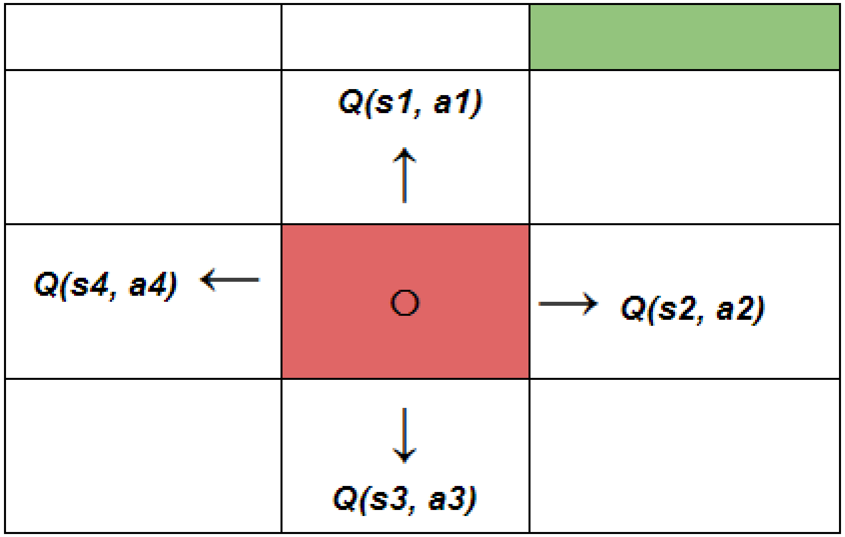
# L7.2 Value-based RL

Zonghua Gu 2021



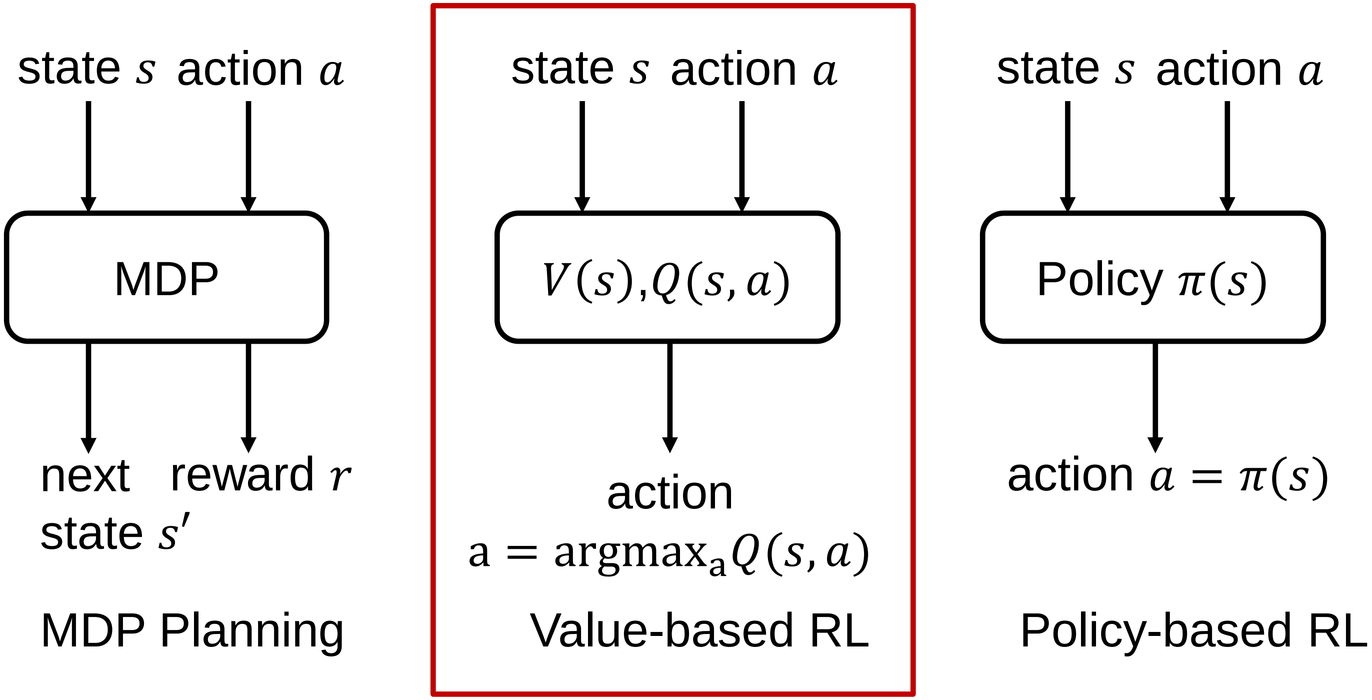
Acknowledgement: slides based on https://www.coursera.org/specializations/reinforcement-learning
And textbook by Sutton and Barto http://incompleteideas.net/book/the-book-2nd.html

Important

# The Big Picture

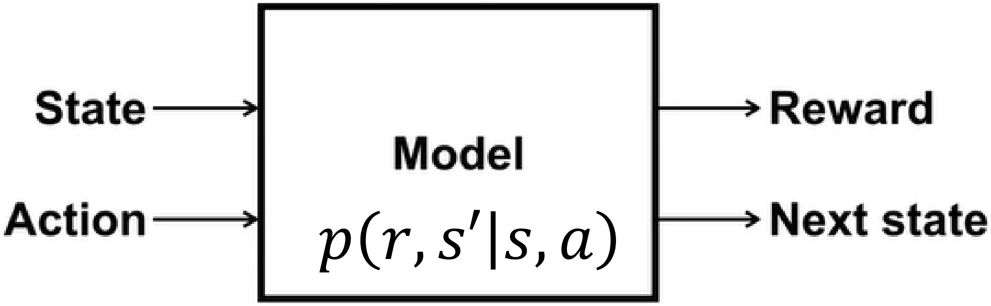
| Problem | Bellman Equation | Algo (known MDP) | Algo (unknown MDP, sample-based) |
|---|---|---|---|
| Prediction (compute $v_\pi(s)$) | Bellman Exp. Equation for $v$ | Policy Evaluation (PE) | MC Prediction, TD Learning (on-policy) |
| Control (compute $v_\pi(s)$, then $\pi(s) = \underset{a}{\operatorname{argmax}}\, q(s,a)$) for known MDP | Bellman Exp. Equation for $v$ + Greedy Policy Improvement (GPI) | Policy Iteration (PI=PE+GPI) | Cannot do GPI, since cannot get $Q(s,a)$ from $V(s)$ without MDP |
| Control (compute $v_*(s)$, then $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, q_*(s,a)$) for known MDP | Bellman Opt. Equation for $v$ | Value Iteration (VI) (a form of Generalized PI) | Cannot compute $V^*(s)$ w. sample-based method due to $\underset{a}{\max}$ in front; cannot get $Q(s,a)$ from $V(s)$ without MDP |
| Control (compute $q_*(s,a)$, then $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, q_*(s,a)$) | Bellman Opt. Equation for $q$ | Q Value Iteration (QVI) | Sarsa (on-policy)<br>Q Learning, Expected Sarsa (off-policy) |

# Value-based RL



# Reinforcement Learning

- Recall: an MDP consists of:
  - Set of states $S$
  - Start state $s_0$
  - Set of actions $A$
  - Transitions and rewards $p(r, s'|s, a)$ (w. discount $\gamma$)
- But now the model $p(r, s'|s, a)$ is unknown
  - Unknown reward $r$ and next state $s'$, denoted as state transition $(s, a, r, s')$, if agent takes action $a$ in state $s$.
  - Agent must learn the optimal policy $\pi(a|s)$ by trial-and-error.



State

Action

Model

$p(r, s'|s, a)$

Reward

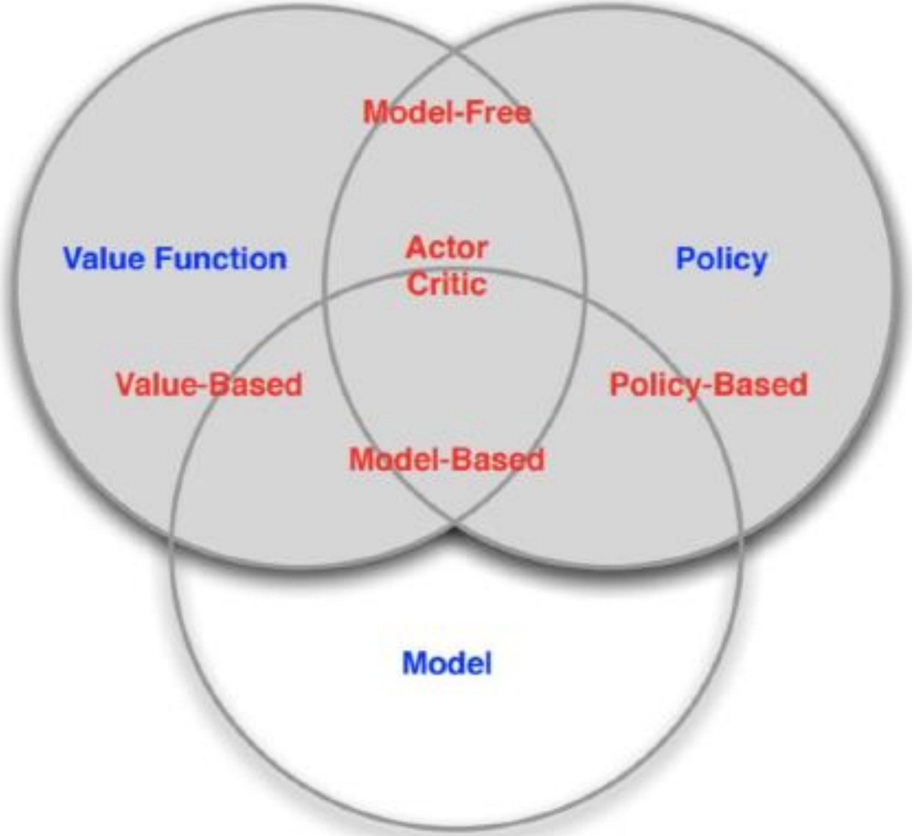
Next state

# Characteristics of RL

- There is no supervisor, only a reward signal (may be sparse)
- Feedback is delayed, not instantaneous
- Sequential, non i.i.d data
  - Agent's actions affect the subsequent data it receives

# Model-based vs. Model-Free RL

- Model-based RL
  - Learn MDP model $p(r, s'|s, a)$ then use Value Iteration or Policy Iteration to solve for the optimal value function and policy
- Model-free
  - Learn the optimal value function and/or policy directly without learning the MDP



# Model-Based vs. Model-Free by Analogy

Goal: Compute expected age of a group of $M$ students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [$a_1$, $a_2$, … $a_N$]

Unknown P(A): "Model-Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model-Free"

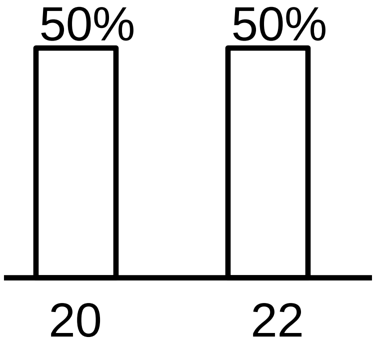
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because samples appear with the right frequencies.

- If we have the model $P(A)$ (probability/percentage of students with a certain age $a$), we can compute expected (average) age by a weighted sum. But without $P(A)$:
  - Model-based: randomly sample $N < M$ students and asking for their ages, in order to build an estimated model $\hat{P}(A)$ ($num(a)$ is the number of sampled students with age $a$.). Then use $\hat{P}(A)$ to compute the expected age.
  - Model-free: randomly sample $N < M$ students and ask for their ages, then compute their expected age directly.

# Model-Based vs. Model-Free Example

- We want to compute expected age of a group of $M$ students. We use random variable $A$ to denote student age. If we knew the group consists of two age groups, 20 and 22, with ground truth distribution $\mathrm{P}(\mathrm{a}=20)=\mathrm{P}(\mathrm{a}=22)=.5$. We can then compute expectation of $A$ as $E[A]=\sum_a P(a)\cdot a=.5\cdot 20+.5\cdot 22=21$. But the ground truth distribution is unknown in general
  - Model−based: randomly sample $N<M$ students to build an estimated model $\hat{P}(A)$, e.g., $\widehat{\mathrm{P}}(\mathrm{a}=20)=.6, \widehat{\mathrm{P}}(\mathrm{a}=22)=.4$ (different from ground truth distribution of $.5{:}.5$). Then use $\hat{P}(A)$ to compute the expected age $E[A]\approx\sum_a \hat{P}(a)\cdot a=.6\cdot 20+.4\cdot 22=20.8$
  - Model-free: randomly sample $N<M$ students, and compute their expected age directly, e.g., we sample 5 students with ages $(20,20,22,22,20)$, then $E[A]\approx \frac{1}{N}\sum_i a_i=\frac{1}{5}(20+20+22+22+20)=20.8$.
- Analogously for RL:
  - Given sufficient samples, both model-based RL and model-free RL should give the same optimal solution. In practice, model-based RL is typically more sample efficient than model-free RL.



Ground truth distribution $P(A)$

# Outline

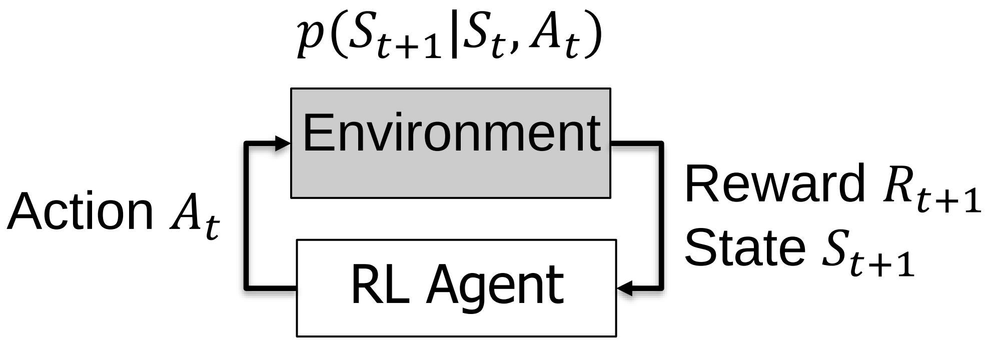
- Monte Carlo Methods
- TD-Learning
- Sarsa & Q-Learning
- Function Approximation

# Outline: Monte Carlo Methods

- MC prediction
- On-policy MC control
  - Exploring Starts
  - $\epsilon$-Soft
- Off-policy MC Prediction via Importance Sampling
- Off-policy MC control (omitted)

# Recall: Definitions of Terms

- Return (cumulative discounted reward) at time $t$: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$
  - At each step $t \in [0, T-1]$, agent takes an action $A_t$ in state $S_t$; at step $t+1$, agent receives a reward $R_{t+1}$ and transitions into the next state $S_{t+1}$
  - We assume episodic tasks, and this specific episode has length of $T$ steps. ($T = \infty$ for continuous tasks)
- State Value Function: expected return under policy $\pi$: $v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s]$
- Action Value Function: expected return from taking action $a$, then follow policy $\pi$: $q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$
- The RL problem: find the optimal policy that maximizes the return from each state



$p(S_{t+1} | S_t, A_t)$

Environment

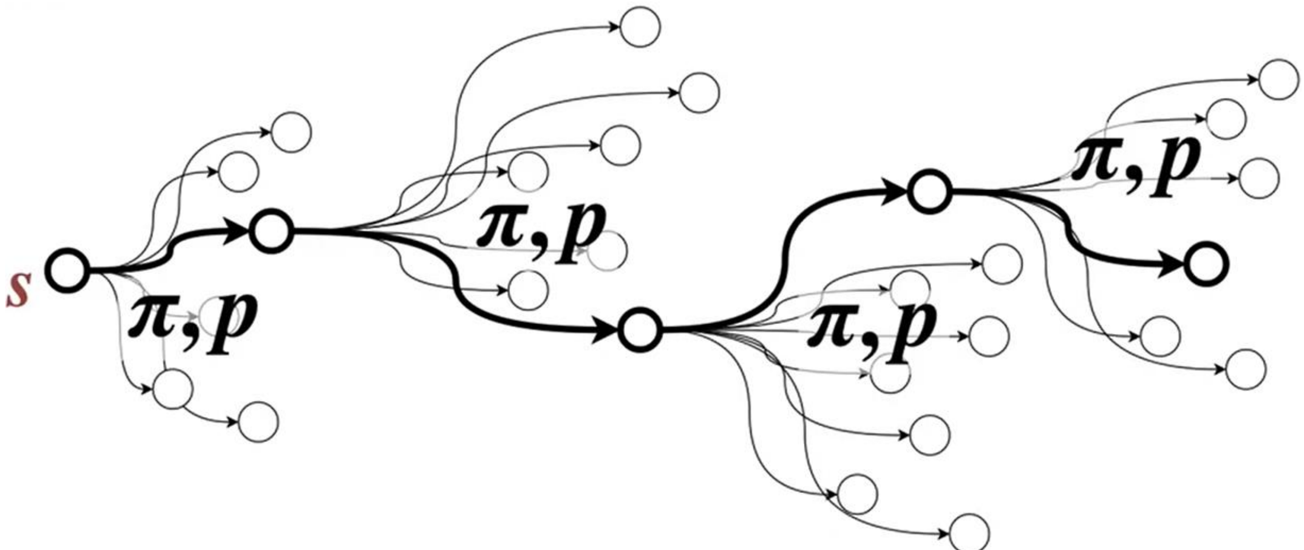
Action $A_t$

Reward $R_{t+1}$
State $S_{t+1}$

RL Agent

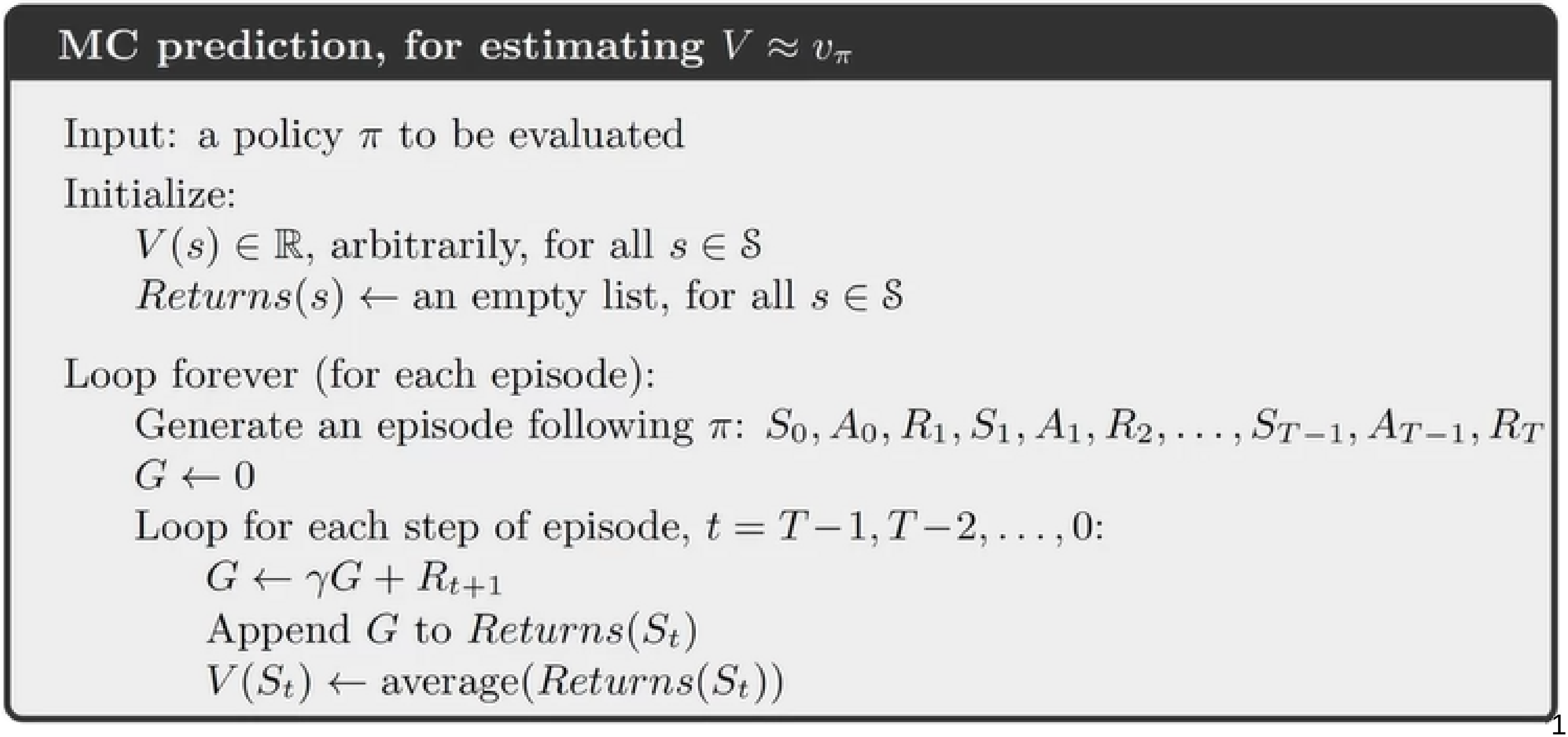
# Monte Carlo (MC) Prediction (Policy Evaluation)

- Collect episodes/trajectories under policy $\pi$; After each episode, compute return $G_t$ for each state $S_t$ encountered in the episode (either first-visit or every-visit); estimate expected return $v_\pi(s)$ with empirical mean return by averaging over all episodes.



# MC Prediction Details

- State $s$ may be visited multiple times in the same episode
  - First-visit MC method estimates $v_\pi(s)$ as the average of the returns following first visits to $s$ in each episode,
  - Every-visit MC method estimates $v_\pi(s)$ as the average of the returns following all visits to $s$ in each episode, computed by going backwards from last non-terminal $S_{T-1}$ (show below),
- For every-visit MC, instead of keeping all the sampled returns in a list, we can incrementally update $v_\pi(s)$ as a moving average with learning rate (step size) $\alpha$ (more recent visits are given higher weight): $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$, where $\alpha$ can be a constant, or be typically reduced gradually until convergence.

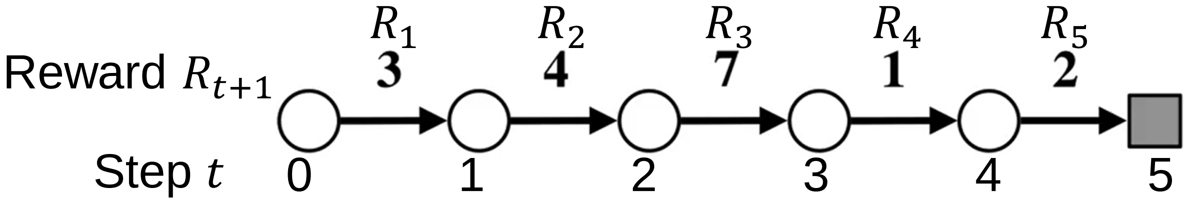


**MC prediction, for estimating $V \approx v_\pi$**

```
Input: a policy $\pi$ 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 $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Append $G$ to $Returns(S_t)$
        $V(S_t) \leftarrow$ average$(Returns(S_t))$
```

# Example: Computing Returns for One Episode

- Working backward is more efficient than working forward as it avoids redundant computations.



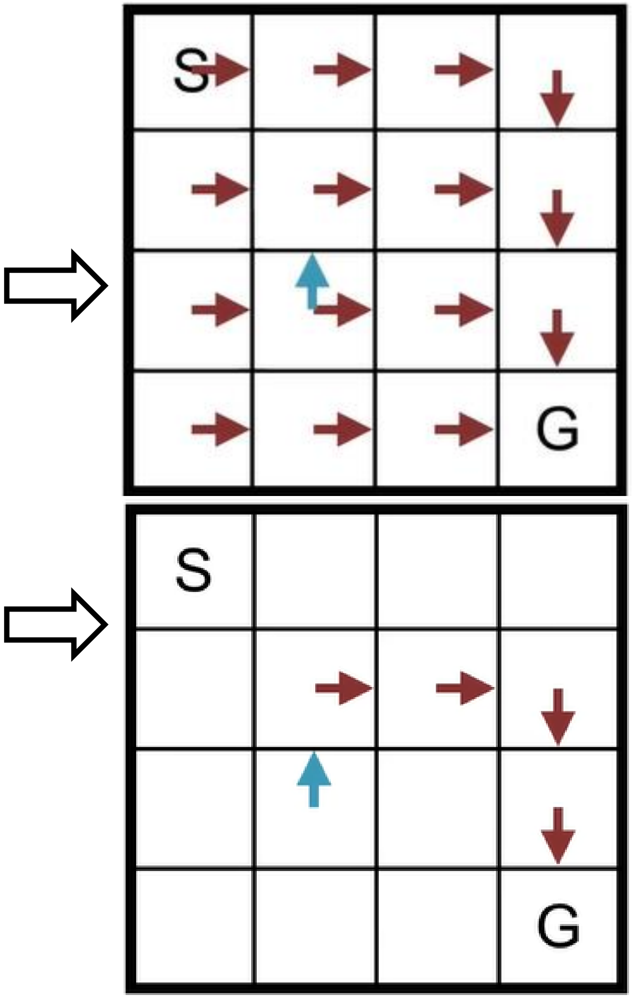
$\gamma = 0.5$

$$
\begin{aligned}
G_0 &= R_1 + \gamma G_1 \\
G_1 &= R_2 + \gamma G_2 \\
G_2 &= R_3 + \gamma G_3 \\
G_3 &= R_4 + \gamma G_4 \\
G_4 &= R_5 + \gamma G_5 \\
G_5 &= 0
\end{aligned}
$$

$$
\begin{aligned}
G_0 &= R_1 + \gamma\, R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \gamma^4 R_5 = 7 \\
G_1 &= R_2 + \gamma\, R_3 + \gamma^2 R_4 + \gamma^3 R_5 = 8 \\
G_2 &= R_3 + \gamma\, R_4 + \gamma^2 R_5 = 8 \\
G_3 &= R_4 + \gamma\, R_5 = 2 \\
G_4 &= R_5 = 2 \\
G_5 &= 0
\end{aligned}
$$

# Exploration in MC Prediction for $q_\pi(s,a)$

- For any policy (det or sto): exploring starts
  - Specify that the episodes start in $(s_0, a_0)$, and that every possible $(s, a)$ has a nonzero probability of being selected as the start.
  - From $(s_1, a_1)$ follow policy $\pi$ for action selection
    - $s_0, a_0, s_1, a_1, s_2, a_2, \ldots$
  - May not be always feasible for complex environments.
- For stochastic policy
  - $\epsilon$-greedy (or $\epsilon$-soft)



# Outline: Monte Carlo Methods

- MC prediction
- On-policy MC control
  - Exploring Starts
  - $\epsilon$-Soft
- Off-policy MC Prediction via Importance Sampling
- Off-policy MC control (omitted)

# Recall: Policy Iteration for Known MDP

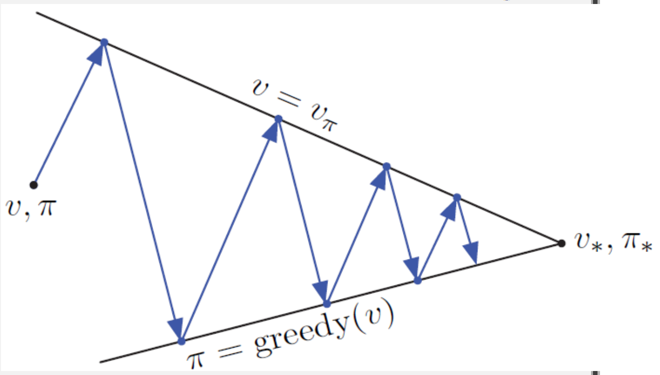
**Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$**

```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

Repeat until policy converges:

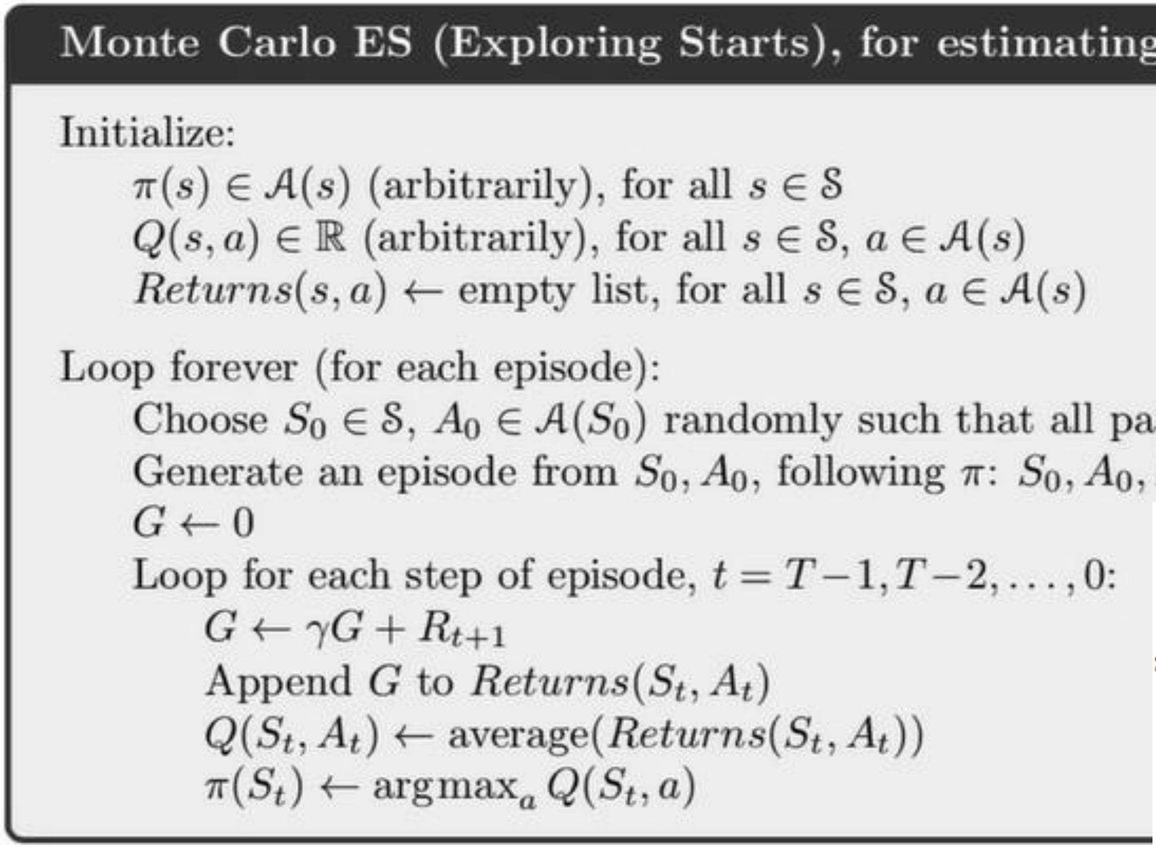
Policy Evaluation: Estimate state value function $v_\pi$ for some fixed policy $\pi$ with Iterative Policy Evaluation (or solving linear equations).

Policy Improvement: generate new policy based on the newly estimated $v_\pi$: $\pi = \text{greedy}(v_\pi)$.



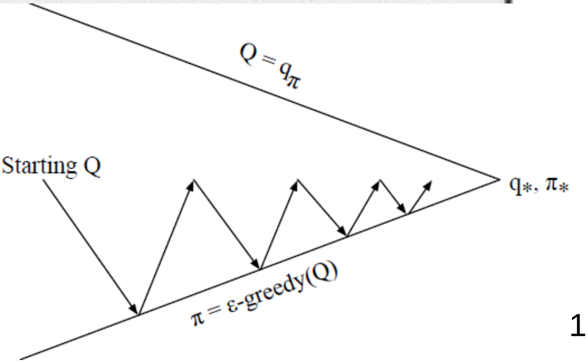
# Generalized Policy Iteration with MC ES (Exploring Starts)

- GPI w. MC ES: After each episode, the observed returns are used for policy evaluation, and then the policy is improved at all the states visited in the episode.



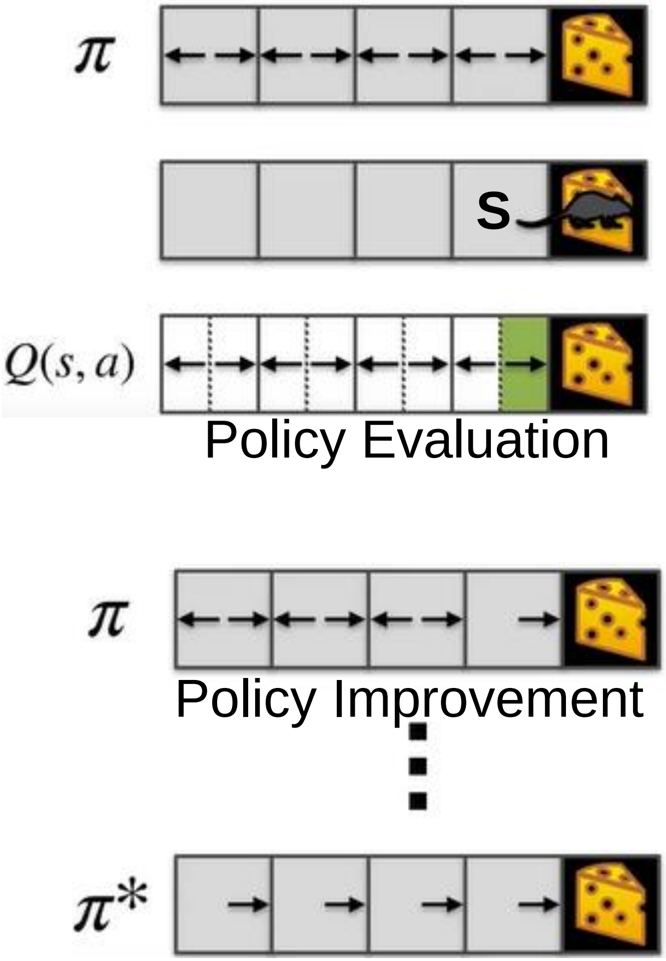
**Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$**

```
Initialize:
    $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability $> 0$
    Generate an episode from $S_0, A_0$, following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Append $G$ to $Returns(S_t, A_t)$
        $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
        $\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$
```



# GPI w. MC Example

- Agent initially has uniform random policy of going left or right. In each episode, it wanders around until it hits the goal and gets a reward, then $Q(s,a)$ for all the states visited in the episode are updated, and policy is improved based on $\underset{a}{\operatorname{argmax}}\, Q(s,a)$.
  - In the fig, agent starts from state **S**, and happens to go right to hit the goal, then only policy of state **S** is changed. In general, policies of all states visited in the episode are updated.
- After sufficient exploration, all states' policies may converge to the optimal policy $\pi^*$.
- If some states are not explored enough, e.g., the leftmost state is never visited, or they are visited only a few times, and the updates to $Q(s,a)$ is not enough to overcome badly initialized $Q_{init}(s,a)$ values, then their policies may not be optimal.



Policy Evaluation

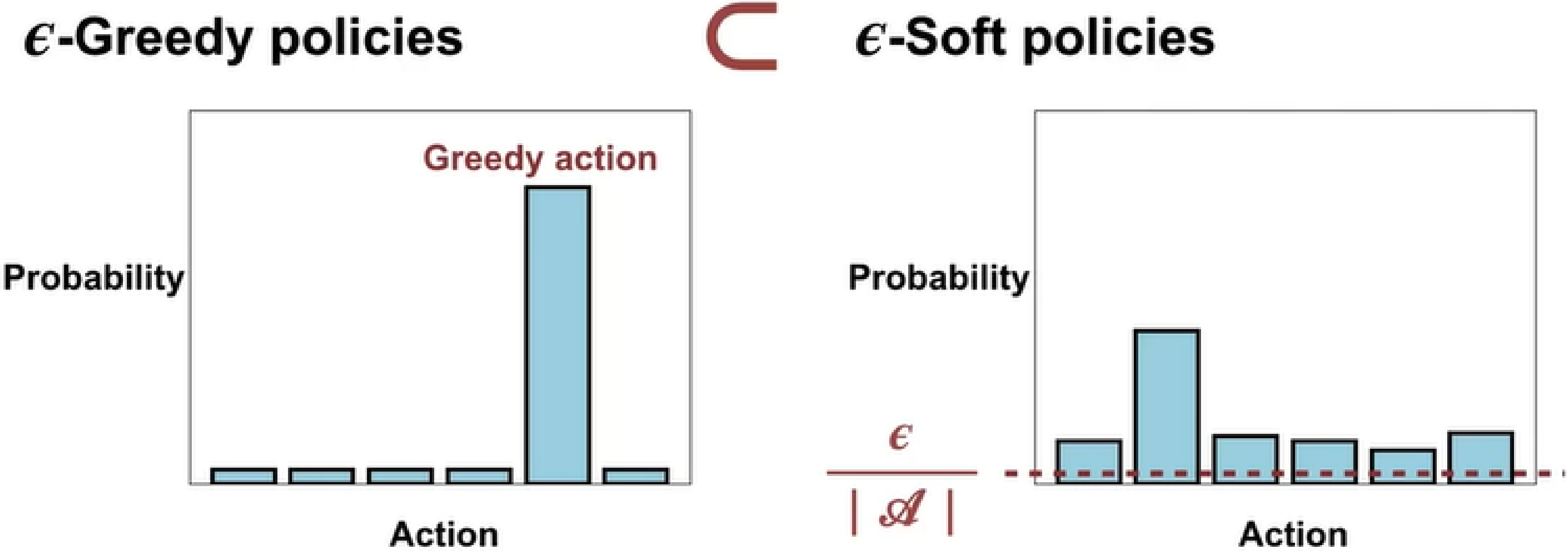
Policy Improvement

# Exploration-Exploitation Dilemma

- The agent has to exploit what it has already experienced in order to obtain reward, but it also has to explore in order to make better action selections in the future.
  - Exploitation: to obtain a lot of reward, the agent must prefer actions that it has tried in the past and found to be effective in producing reward.
  - Exploration: to discover such actions, it has to try actions that it has not selected before.

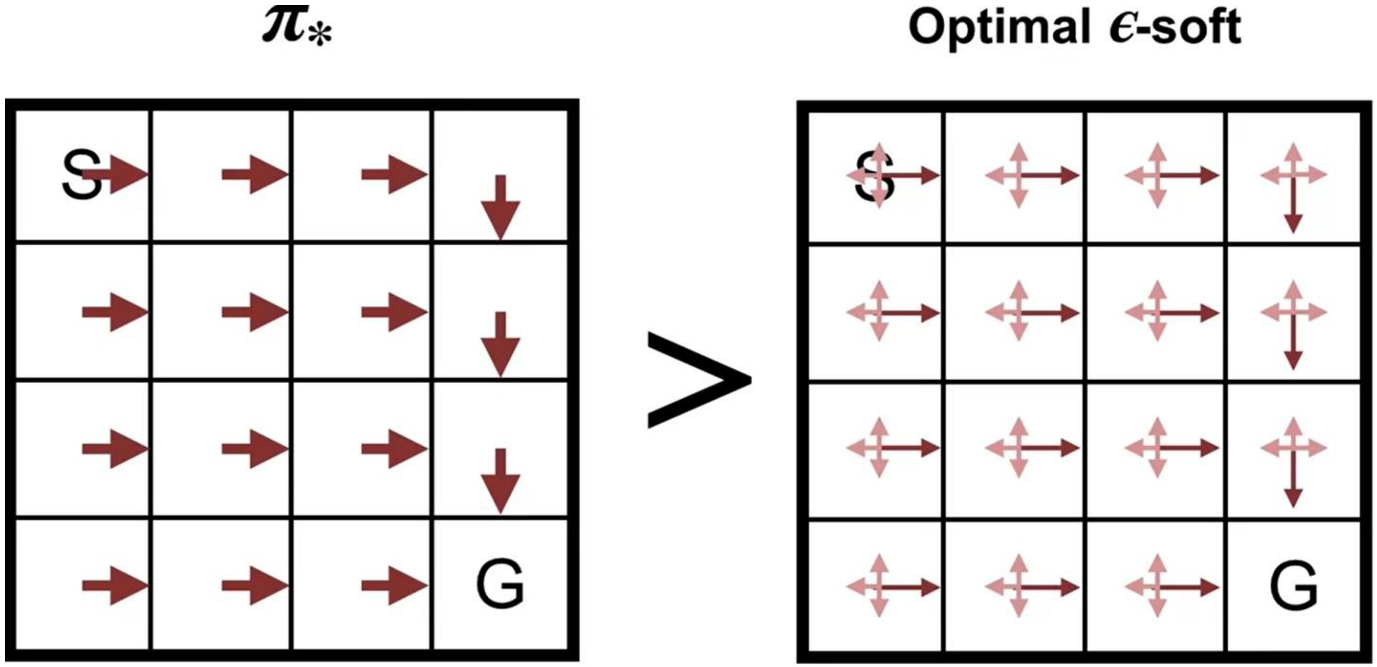
# $\epsilon$-Greedy as One Type Of $\epsilon$-Soft

- $\epsilon$-greedy policy: select a random action w. prob $\epsilon$ (exploration); select the greedy action $\operatorname*{argmax}_a Q(s,a)$ with prob $1-\epsilon$ (exploitation)
  - With $|\mathcal{A}(s)|$ possible actions in state $s$, select each non-greedy action w. prob $\frac{\epsilon}{|\mathcal{A}(s)|}$; the greedy action w. prob $1-\epsilon+\frac{\epsilon}{|\mathcal{A}(s)|}$
- $\epsilon$-soft policy: $\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$ for all $(s,a)$, and for some $\epsilon > 0$
- $\epsilon$-greedy policy is a special case of $\epsilon$-soft policy

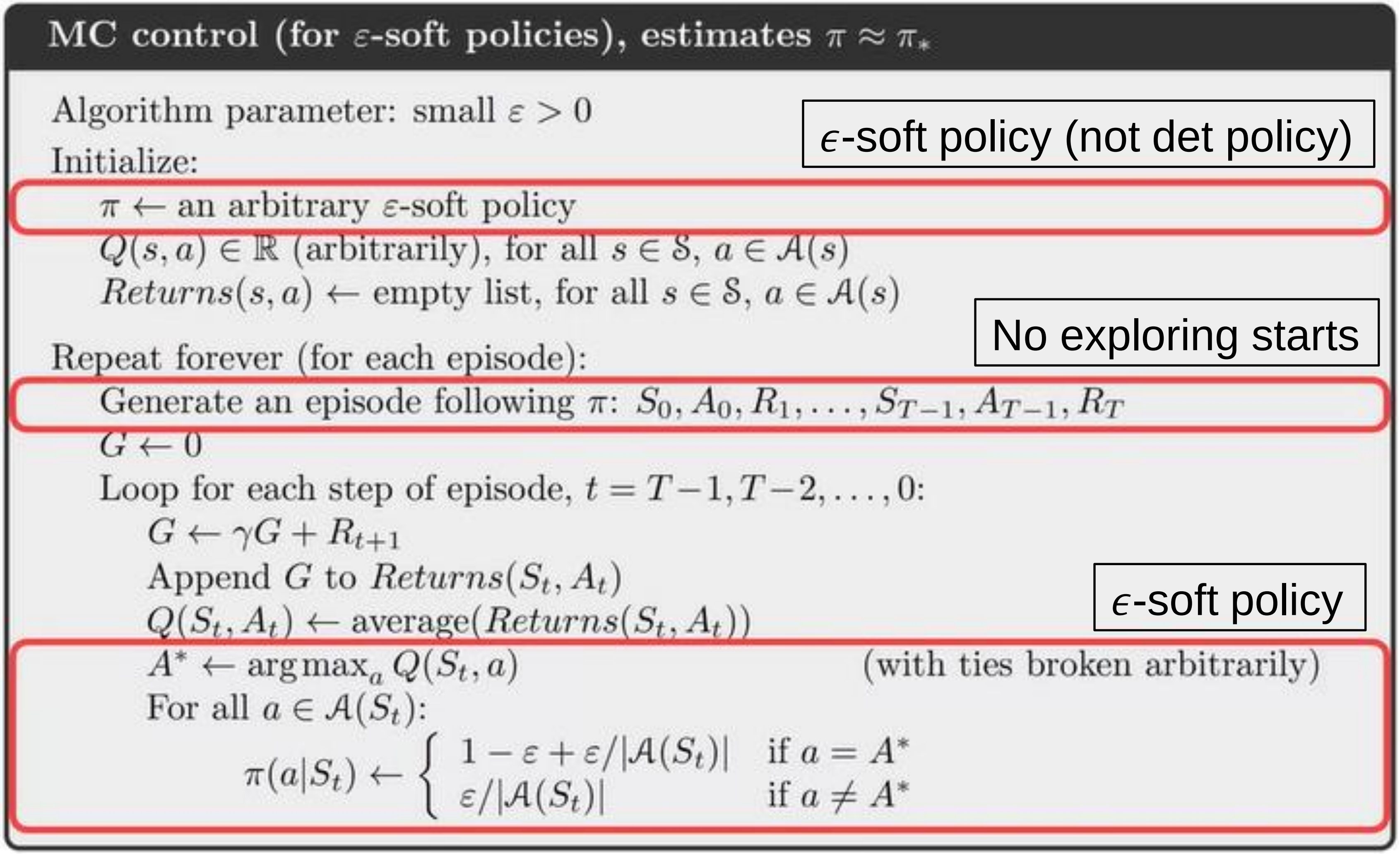


# Optimal $\epsilon$-Soft Policy

- The optimal $\epsilon$-soft policy is the policy with the highest value in each state among all $\epsilon$-soft policies. It performs worse than the optimal greedy deterministic policy $\pi_*$ in general.
- But it often performs reasonably well, and avoids exploring starts.



# Generalized Policy Iteration with MC $\epsilon$-soft



**MC control (for $\varepsilon$-soft policies), estimates $\pi \approx \pi_*$**

```
Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-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 $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Append $G$ to $Returns(S_t, A_t)$
        $Q(S_t, A_t) \leftarrow$ average($Returns(S_t, A_t)$)
        $A^* \leftarrow \arg\max_a Q(S_t, a)$                    (with ties broken arbitrarily)
        For all $a \in \mathcal{A}(S_t)$:
            $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

$\epsilon$-soft policy (not det policy)

No exploring starts

$\epsilon$-soft policy

# Outline: Monte Carlo Methods

- MC prediction
- On-policy MC control
  - Exploring Starts
  - $\epsilon$-Soft
- Off-policy MC Prediction via Importance Sampling
- Off-policy MC control (omitted)

# On-Policy and Off-Policy

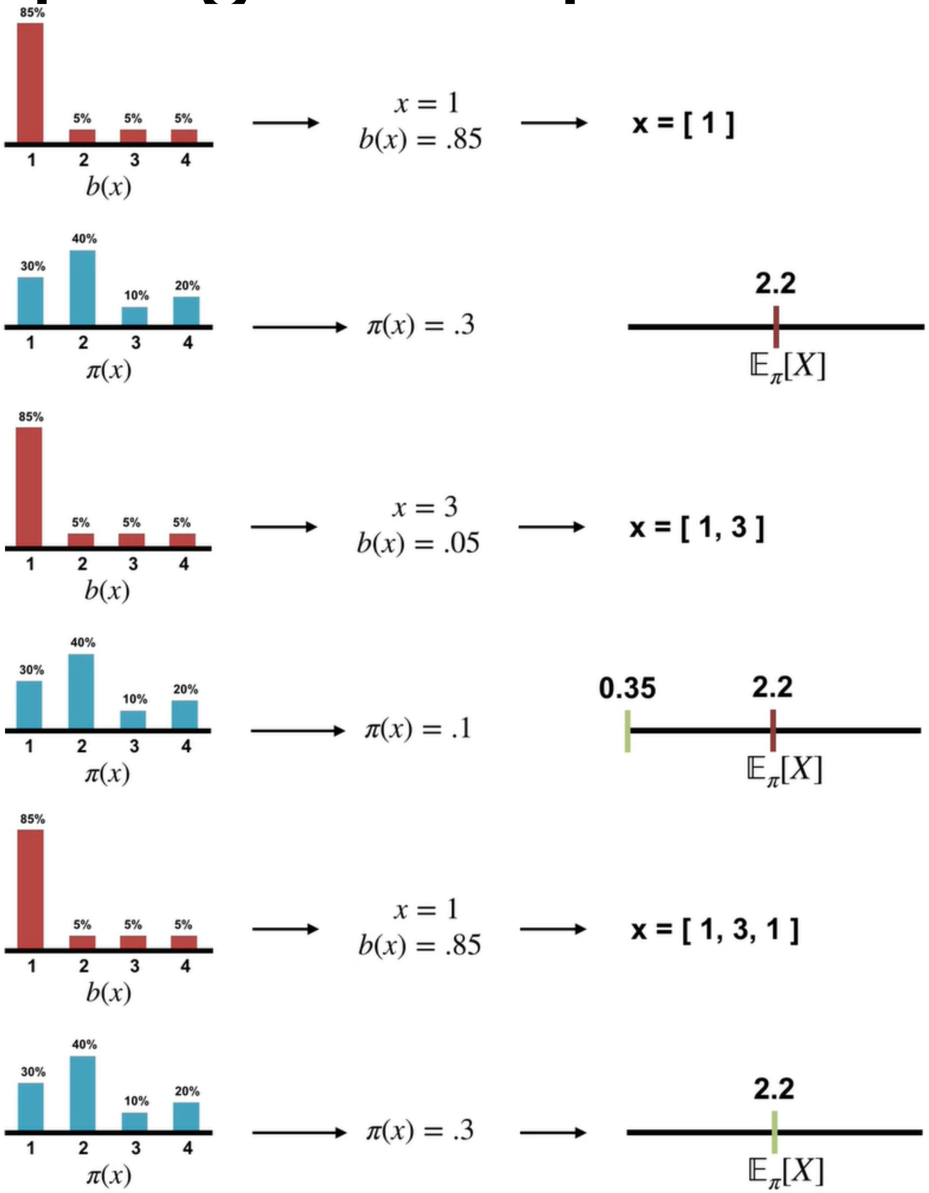
- Off-Policy: Improve and evaluate a different target policy $\pi(a|s)$ from the behavior policy $b(a|s)$ that is used to select actions.
  - Behavior policy may be more random, and more exploratory/adventurous than target policy, to facilitate exploration
  - $\pi(a|s) > 0 \Rightarrow b(a|s) > 0$ ($b(a|s)$ must cover $\pi(a|s)$. If $a$ is possible in target policy, it must be possible in behavior policy. Otherwise agent will never experience $(s, a)$
  - “Look over someone's shoulder”
- On-Policy: Improve and evaluate the behavior policy $b(a|s)$ that is being used to select actions.
  - $b(a|s) == \pi(a|s)$
  - “Learn on the job”

# Importance Sampling

- We want to estimate $\mathbb{E}_\pi[X]$, expected value of random var $X$ with distribution $\pi$, by sampling from another distribution $x \sim \rho$.
  - Capital letter ($X$) denotes a random variable; lower-case letter ($x$) denotes a sampled value of the random var $X$.
- $\mathbb{E}_\pi[X] \doteq \sum_{x \in X} \pi(x)x = \sum_{x \in X} b(x)\rho(x)x = \mathbb{E}_b[\rho(X)X] \approx \frac{1}{n}\sum_{i=1}^{n} x_i\rho(x_i)$
  - $\rho(x_i) = \frac{\pi(x_i)}{b(x_i)}, x_i \sim b$
- We compute $\mathbb{E}_\pi[X]$ by sampling from $b$, then performing weighted average with weight $\rho(x)$.

# Importance Sampling Example

- $\mathbb{E}_\pi[X] = .3\cdot 1 + .4\cdot 2 + .1\cdot 3 + .2\cdot 4 = 2.2$
- 1st sample from $b(x)$: get $x = 1$ w. prob $0.85$
  - $\mathbb{E}_b[X] = \frac{1}{1}\sum_1^1 x\rho(x) = 1\times\frac{.3}{.85} = 0.35$
- 2nd sample from $b(x)$: get $x = 3$ w. prob $0.05$
  - $\mathbb{E}_b[X] = \frac{1}{2}\sum_1^2 x\rho(x) = \frac{1}{2}\left(1\times\frac{.3}{.85} + 3\times\frac{.1}{.05}\right) \approx 3.18$
- 3rd sample from $b(x)$: get $x = 1$ w. prob $0.85$
  - $\mathbb{E}_b[X] = \frac{1}{3}\sum_1^3 x\rho(x) = \frac{1}{3}\left(1\times\frac{.3}{.85} + 3\times\frac{.1}{.05} + 1\times\frac{.3}{.85}\right) \approx 2.24$

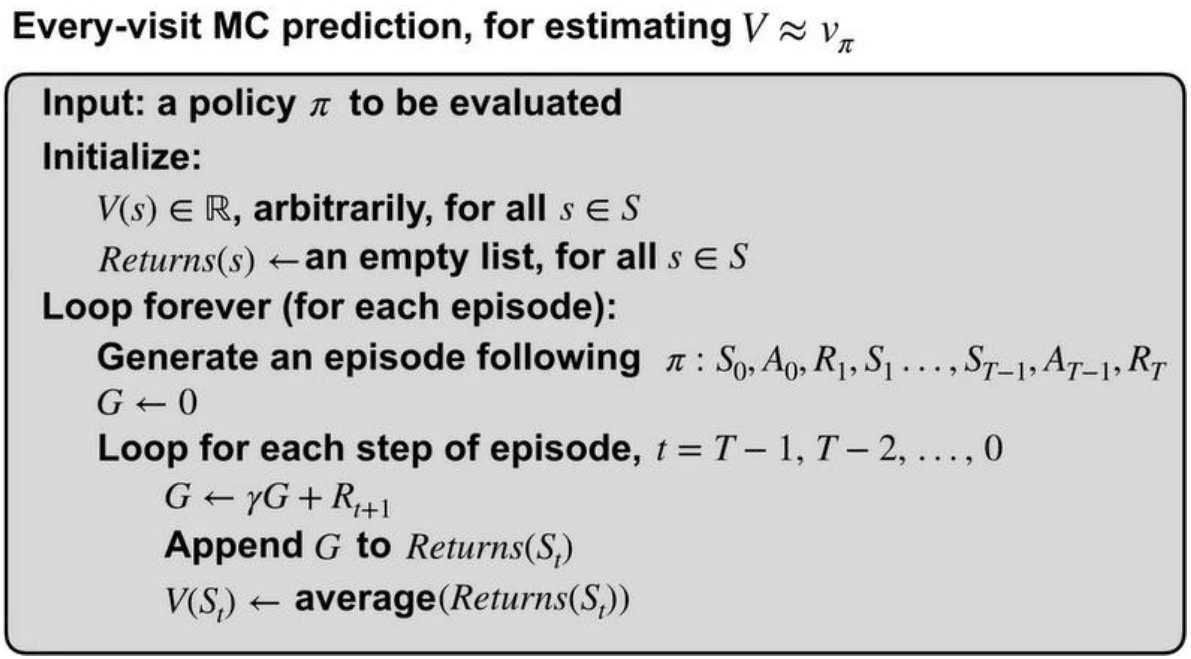


# Computing Weighting Factors

- Prob of each off-policy trajectory under behavior policy $b$: $\mathbb{P}(\text{traj under } b) \doteq \mathbb{P}(A_t, S_{t+1}, A_{t+1}, \dots, S_T | S_t, A_{t:T}) = \prod_{k=t}^{T-1} b(A_k|S_k)\, p(S_{k+1}|S_k, A_k)$
  - Due to the Markov property.
- $\rho_{t:T-1} \doteq \frac{\mathbb{P}(\text{traj under } \pi)}{\mathbb{P}(\text{traj under } b)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$
  - Work backwards to compute incrementally $W_1 \leftarrow \rho_{T-1}; W_2 \leftarrow W_1\rho_{T-2}; W_3 \leftarrow W_2\rho_{T-3}$
- Value function update w. importance sampling:
  - $V(S_t) \leftarrow V(S_t) + \alpha(\rho_{t:T-1} G_t - V(S_t))$

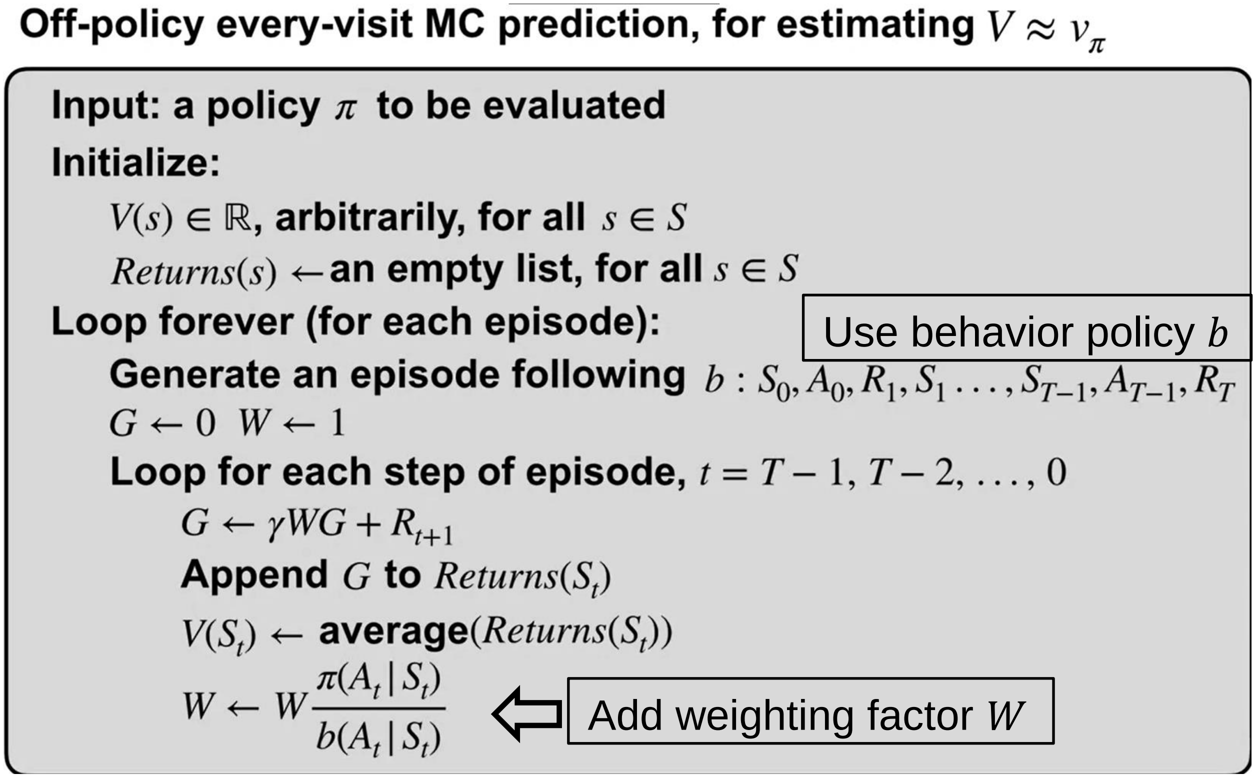
# On-Policy vs. Off-Policy MC Prediction

- Improve and evaluate a different target policy $\pi(a|s)$ from the behavior policy $b(a|s)$ that is used to select actions.



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

```
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in S$
    $Returns(s) \leftarrow$ an empty list, for all $s \in S$
Loop forever (for each episode):
    Generate an episode following $\pi : S_0, A_0, R_1, S_1 \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$
        $G \leftarrow \gamma G + R_{t+1}$
        Append $G$ to $Returns(S_t)$
        $V(S_t) \leftarrow$ average$(Returns(S_t))$
```



Off-policy every-visit MC prediction, for estimating $V \approx v_\pi$

```
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in S$
    $Returns(s) \leftarrow$ an empty list, for all $s \in S$
Loop forever (for each episode):
    Generate an episode following $b : S_0, A_0, R_1, S_1 \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$  $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$
        $G \leftarrow \gamma W G + R_{t+1}$
        Append $G$ to $Returns(S_t)$
        $V(S_t) \leftarrow$ average$(Returns(S_t))$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
```

Use behavior policy $b$

Add weighting factor $W$

# Outline

- Monte Carlo Methods
- TD-Learning
- Sarsa & Q-Learning
- Function Approximation

# Exponential Moving Average

- The running average update: $\bar{x}_n \leftarrow (1-\alpha)\bar{x}_{n-1} + \alpha x_n = \bar{x}_{n-1} + \alpha(x_n - \bar{x}_{n-1})$
  - Shorthand notation $\bar{x}_n \leftarrow_\alpha x_n$
- Makes recent samples more important (since later ones are more accurate estimates)
  - $$\begin{aligned} \bar{x}_n &= \alpha(x_n + (1-\alpha)(\alpha x_{n-1} + (1-\alpha)(\ldots))) \\ &= \alpha(x_n + (1-\alpha)x_{n-1} + (1-\alpha)^2 x_{n-2} + \cdots) \\ &= \frac{x_n + (1-\alpha)x_{n-1} + (1-\alpha)^2 x_{n-2} + \cdots}{1 + (1-\alpha) + (1-\alpha)^2 + \cdots} \end{aligned}$$
  - Since $\frac{1}{\alpha} = 1 + (1-\alpha) + (1-\alpha)^2 + \cdots$
- Forgets about the past gradually (distant past values are likely to be wrong, esp. for changing env.)
- Decreasing learning rate $\alpha$ gradually can give converging average.

# TD Learning

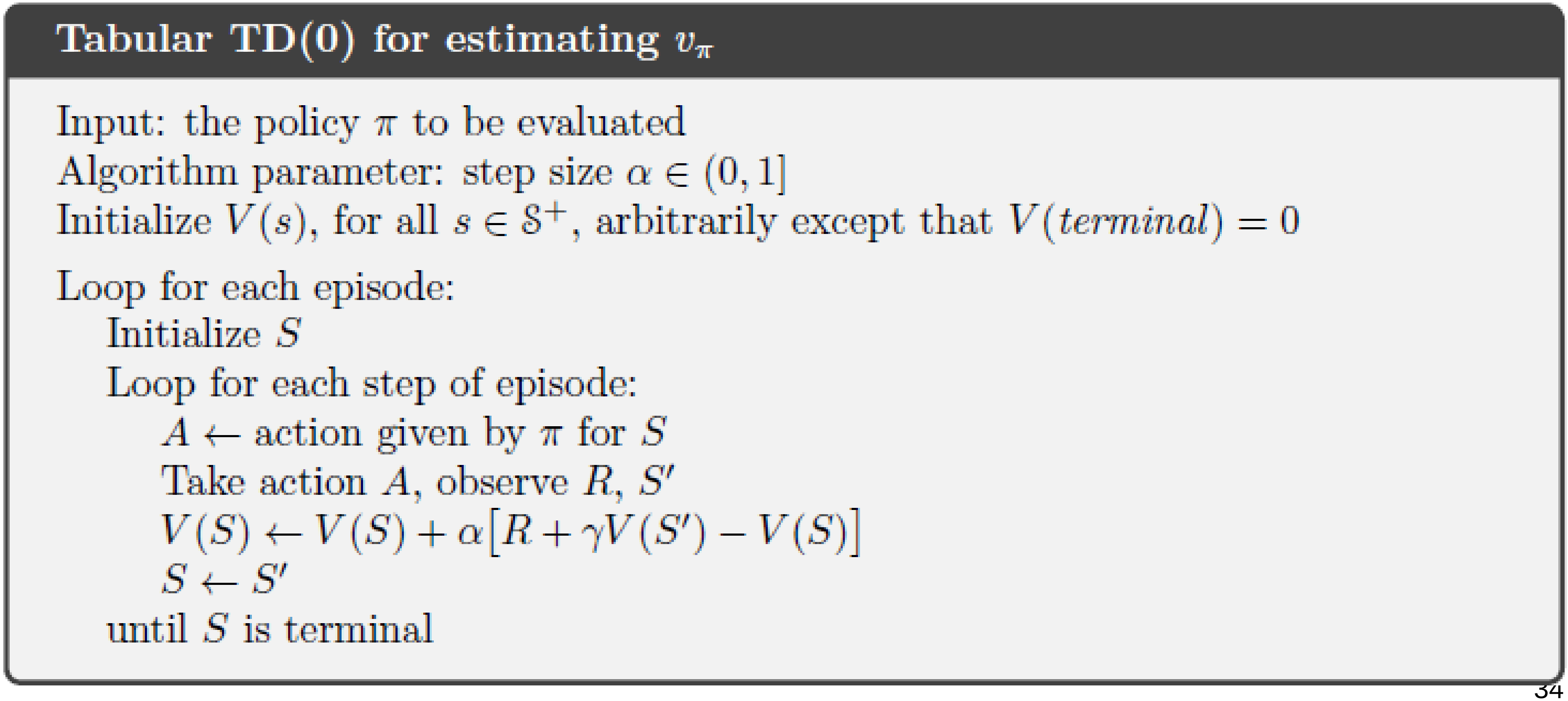
- Recall Bellman Exp. Equation:
  - $v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a) \left[r + \gamma v_\pi(s')\right] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$
  - To solve it with dynamic programming, we need the MDP model $p(r, s'|s, a)$.
- TD Learning: compute $v_\pi(s)$ in model-free way by sampling. At every timestep $t$, update $V(S_t)$ for current state $S_t$:
  - $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
  - TD Target: $R_{t+1} + \gamma V(S_{t+1})$
  - TD Error: $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

# Update Equations: MC vs. TD

- MC (every-visit): After every episode, update $V(S_t)$ for all states encountered in the episode:
  - On-policy MC: $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$
  - Off-policy MC w. Importance Sampling: $V(S_t) \leftarrow V(S_t) + \alpha(\rho_{t:T-1} G_t - V(S_t))$
    - $\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$
- TD: At every timestep $t$, update $V(S_t)$ for current state $S_t$:
  - On-policy TD: $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
  - Off-policy TD w. Importance Sampling: $V(S_t) \leftarrow V(S_t) + \alpha(\rho_t(R_{t+1} + \gamma V(S_{t+1})) - V(S_t))$
    - $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
    - Much lower variance than MC w. Importance Sampling: $V(S_t) \leftarrow V(S_t)$.

# Tabular TD(0)

- MC updates $V(S)$ at the end of each episode.
- TD updates $V(S)$ at every time step.
  - Bootstrapping $V(S)$ from $V(S')$

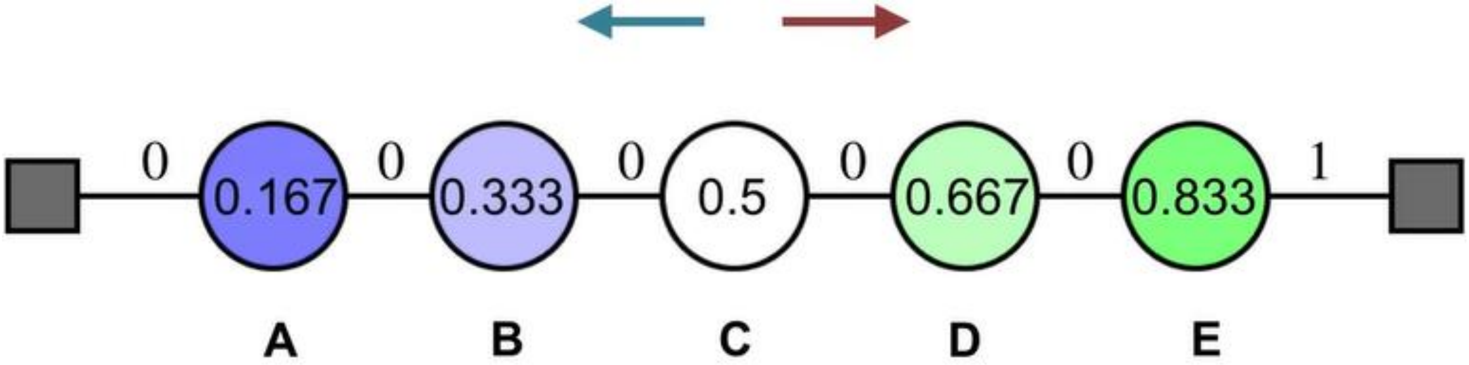


**Tabular TD(0) for estimating $v_\pi$**

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

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ 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 vs. MC: Random Walk Example

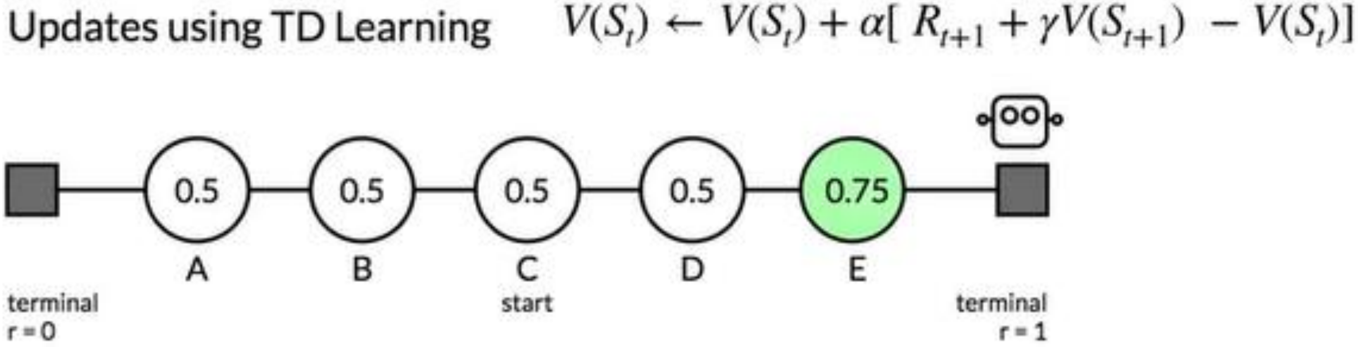
- Agent has uniform random policy, w. equal prob of going left or right at each timestep. Env is deterministic. All episodes start in state $C$. Episodes terminate either on the left or on the right. The reward is $0$ on all transitions except $1$ for terminating on the right. Discount factor $\gamma = 1$: learning rate $\alpha = 0.5$.
- Value of state $\mathrm{V}(s)$ is the probability of terminating on the right when starting from state $s$. For known MDP, they can be computed by Policy Evaluation w. the set of Bellman Exp Equations,w. solution shown in the figure:
  - $V(A) = 0.5V(B)$
  - $V(B) = 0.5V(A) + 0.5V(C)$
  - $V(C) = 0.5V(B) + 0.5V(D)$
  - $V(D) = 0.5V(C) + 0.5V(E)$
  - $V(E) = 0.5V(D) + 0.5 \cdot 1$
- Next, for unknown MDP, we use TD or MC to learn $V(s)$. Initialize all $\mathrm{V}(\cdot) = 0.5$.



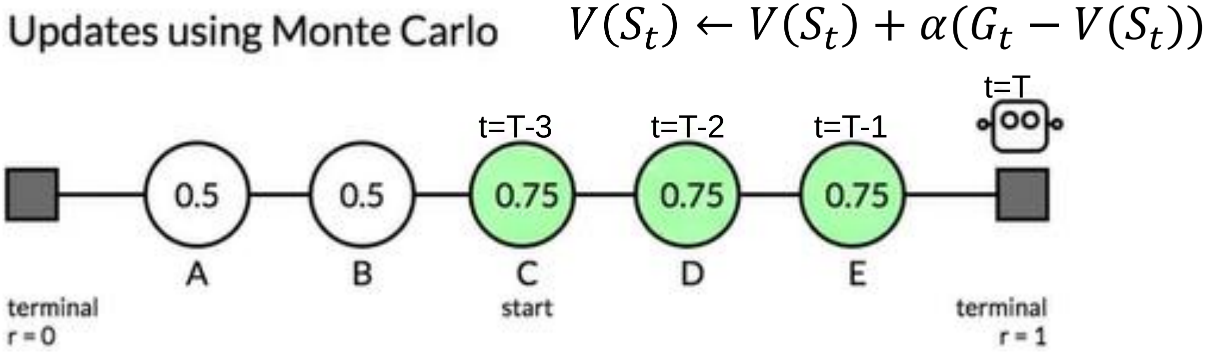
$$\pi(\,.\,|\,s) = 1/2 \;\; \forall s \in \mathcal{S} \qquad\qquad \gamma = 1$$

# Episode1 $(C, r, 0, D, r, 0, E, r, 1, T)$

- TD:
  - $V(C) \leftarrow V(C) + \alpha[R_{t+1} + \gamma V(D) - V(C)] = .5 + .5(0 + .5 - .5) = 0.5$
  - $V(D) \leftarrow V(D) + \alpha[R_{t+1} + \gamma V(E) - V(D)] = .5 + .5(0 + .5 - .5) = 0.5$
  - $V(E) \leftarrow V(E) + \alpha[R_{t+1} + \gamma V(T) - V(E)] = .5 + .5(1 + 0 - .5) = 0.75$
- MC: (every-visit. when used as subscript, $T$ denotes the time instant of reaching the terminal state)
  - $G(E) = R_T = 1, G(D) = R_{T-1} + \gamma G(E) = 0 + 1 \cdot 1 = 1, G(C) = R_{T-2} + \gamma G(D) = 0 + 1 \cdot 1 = 1$
  - $V(C) \leftarrow V(C) + \alpha[G(C) - V(C)] = .5 + .5(1 - .5) = 0.75$
  - $V(D) \leftarrow V(D) + \alpha[G(D) - V(D)] = .5 + .5(1 - .5) = 0.75$
  - $V(E) \leftarrow V(E) + \alpha[G(E) - V(E)] = .5 + .5(1 - .5) = 0.75$



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



Updates using Monte Carlo $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$

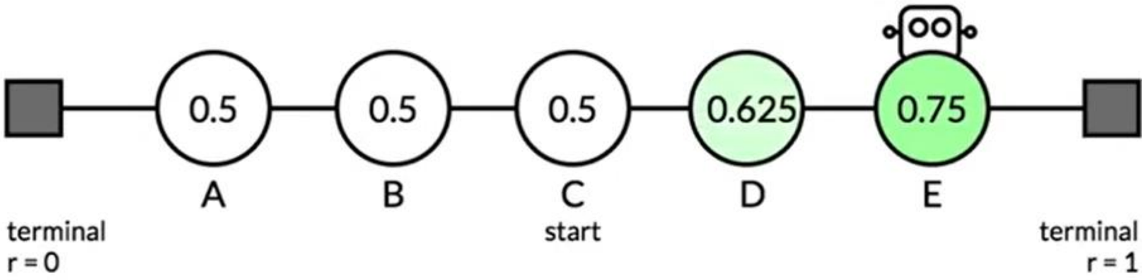
# Quiz on TD vs. MC

- Q: Is the following a more efficient TD update sequence with backward, in-place updates?
  - $V(E) \leftarrow V(E) + \alpha[R_{t+1} + \gamma V(T) - V(E)] = .5 + .5(1 + 0 - .5) = 0.75$
  - $V(D) \leftarrow V(D) + \alpha[R_{t+1} + \gamma V(E) - V(D)] = .5 + .5(0 + .75 - .5) = 0.625$
  - $V(C) \leftarrow V(C) + \alpha[R_{t+1} + \gamma V(D) - V(C)] = .5 + .5(0 + .625 - .5) = 0.5625$
- A: This is not feasible since TD updates $V(S)$ at every time step, and does not keep track of the history of visited states
- Q: Can we use backward, in-place updates for MC?
  - $V(E) \leftarrow V(E) + \alpha[G(E) - V(E)] = .5 + .5(1 - .5) = 0.75$
  - $V(D) \leftarrow V(D) + \alpha[G(D) - V(D)] = .5 + .5(1 - .5) = 0.75$
  - $V(C) \leftarrow V(C) + \alpha[G(C) - V(C)] = .5 + .5(1 - .5) = 0.75$
- A: Yes, but it makes no difference in this example whether you use forward or backward update for $V(S_t)$, which is based on $G(S_t)$, not bootstrapped off $V(S_{t+1})$.
- For every-visit MC using $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$ to update each state's value function, we should use forward update to give more recent visits higher weight, in case the same state is visited multiple times in one episode (refer to p.13 "MC Prediction Details")

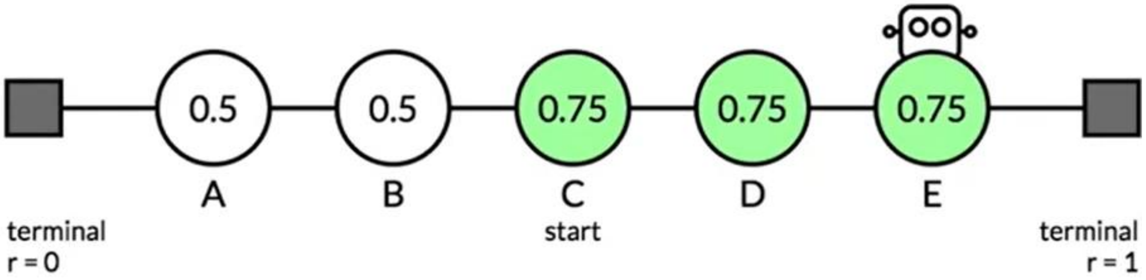
# Episode2 Pt1 $(C, r, 0, D, r, 0, E)$

- TD:
  - $V(C) \leftarrow V(C) + \alpha[R_{t+1} + \gamma V(D) - V(C)] = .5 + .5(0 + .5 - .5) = 0.5$
  - $V(D) \leftarrow V(D) + \alpha[R_{t+1} + \gamma V(E) - V(D)] = .5 + .5(0 + .75 - .5) = 0.625$
- MC:
  - No update since episode has not ended.

Updates using TD Learning

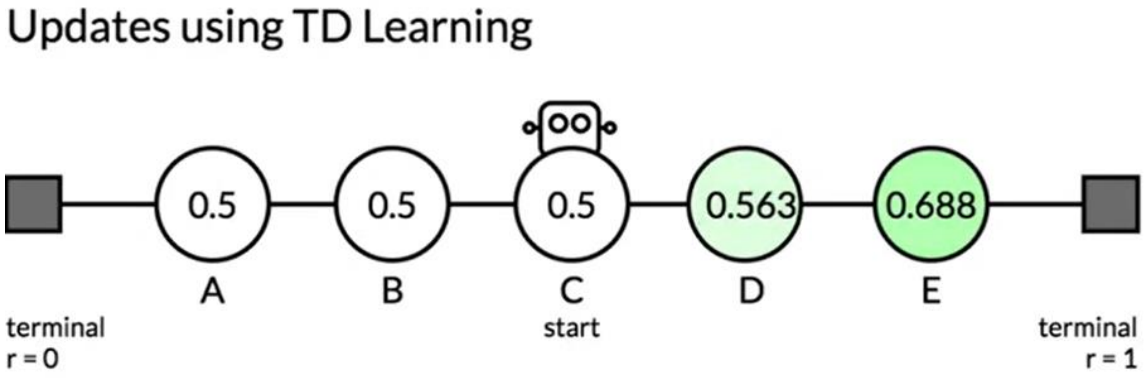


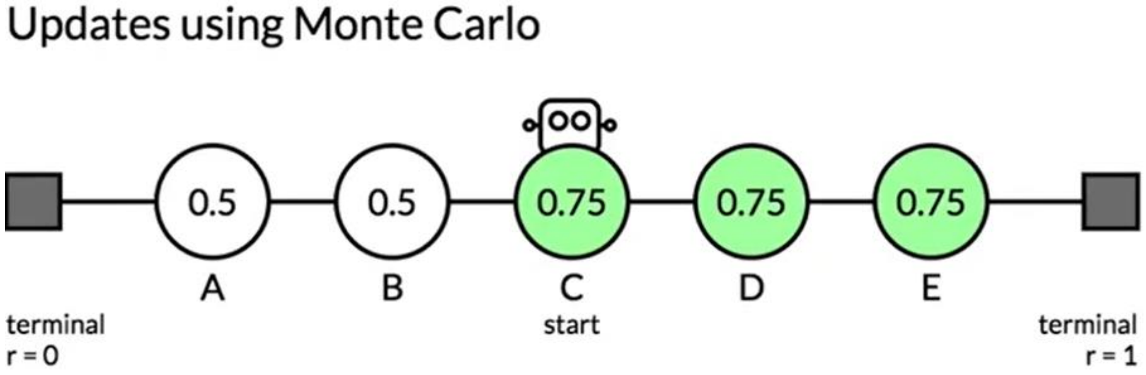
Updates using Monte Carlo



# Episode2 Pt2 $(E, l, 0, D, l, 0, C)$

- TD:
  - $V(E) \leftarrow V(E) + \alpha[R_{t+1} + \gamma V(D) - V(E)] = .75 + .5(0 + .625 - .75) \approx 0.688$
  - $V(D) \leftarrow V(D) + \alpha[R_{t+1} + \gamma V(C) - V(D)] = .625 + .5(0 + .5 - .625) \approx 0.563$
- MC:
  - No update since episode has not ended.

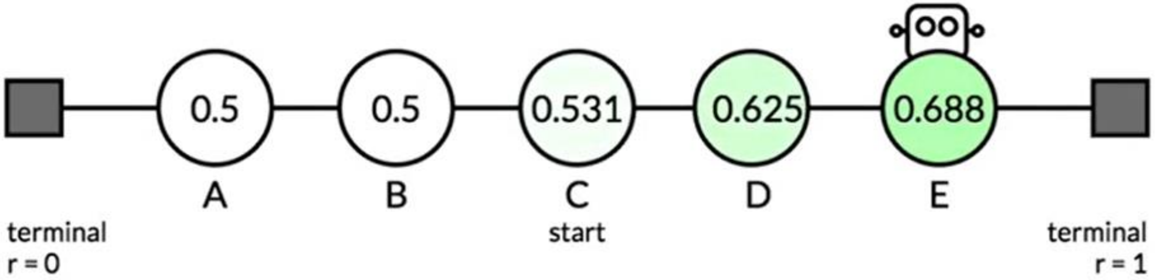

Updates using TD Learning


Updates using Monte Carlo

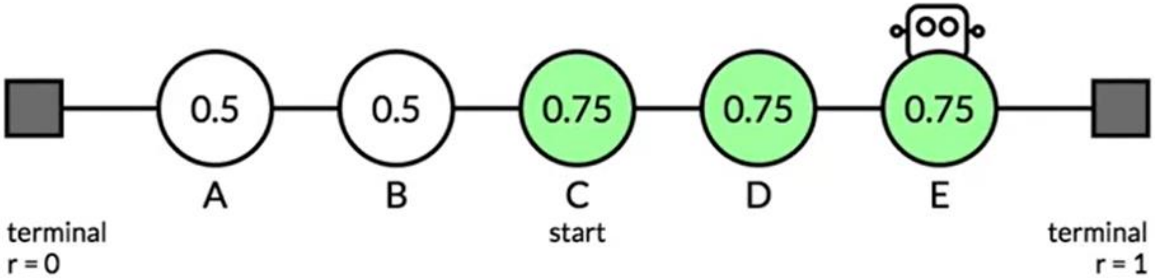
# Episode2 Pt3 $(C, r, 0, D, r, 0, E)$

- TD:
  - $V(C) \leftarrow V(C) + \alpha[R_{t+1} + \gamma V(D) - V(C)] = .5 + .5(0 + .563 - .5) \approx 0.531$
  - $V(D) \leftarrow V(D) + \alpha[R_{t+1} + \gamma V(E) - V(D)] = .563 + .5(0 + .688 - .563) \approx 0.625$
- MC:
  - No update since episode has not ended.

Updates using TD Learning

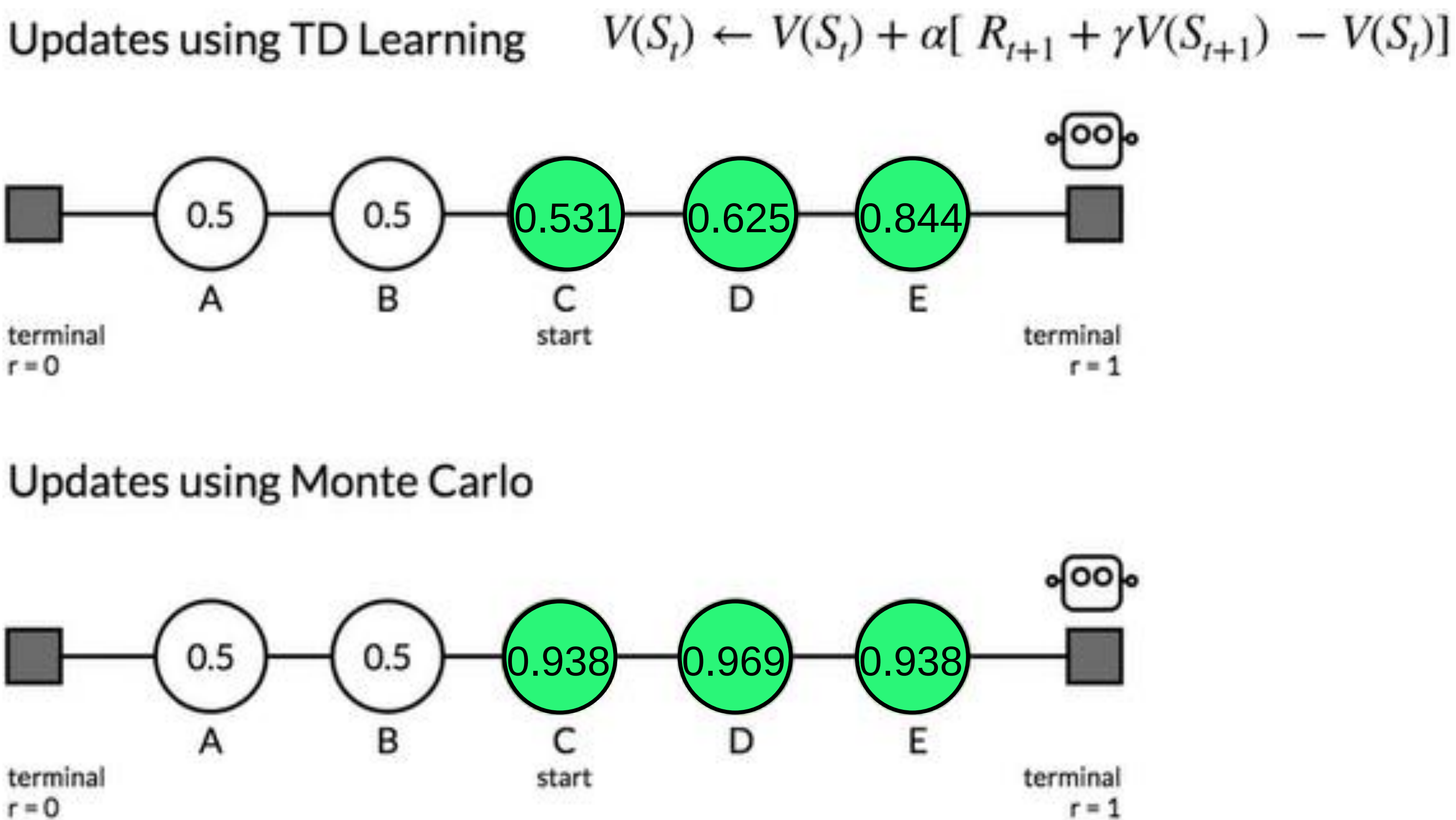


Updates using Monte Carlo



# Episode2 Pt4 $(E, r, 1, T)$

- TD:
  - $V(E) \leftarrow V(E) + \alpha[R_{t+1} + \gamma V(T) - V(E)] = .688 + .5(1 + 0 - .688) = 0.844$

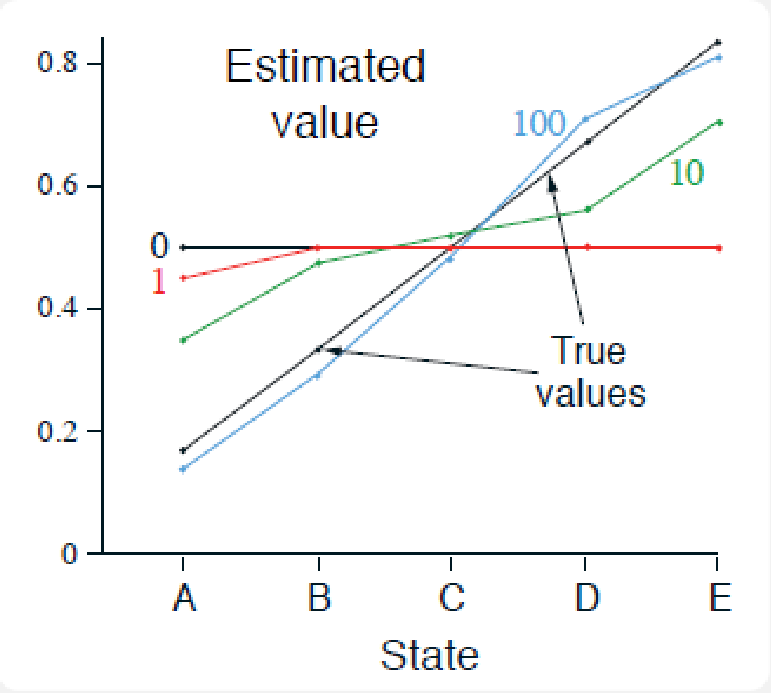

Updates using TD Learning
$V(S_t) \leftarrow V(S_t) + \alpha[\, R_{t+1} + \gamma V(S_{t+1}) \; - V(S_t)]$
0.5
0.5
0.531
0.625
0.844
A
B
C
start
D
E
terminal
r = 0
terminal
r = 1
Updates using Monte Carlo
0.5
0.5
0.938
0.969
0.938
A
B
C
start
D
E
terminal
r = 0
terminal
r = 1

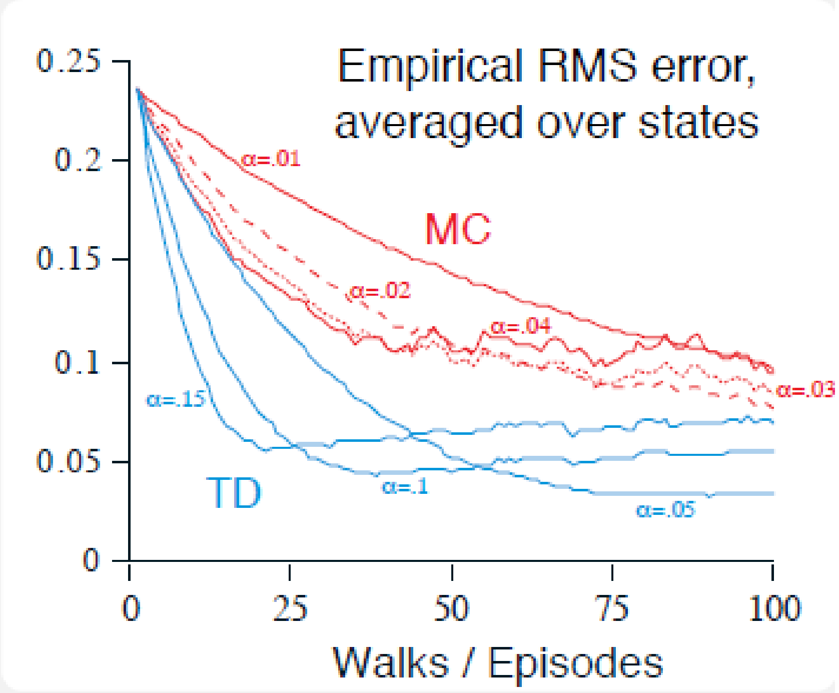
# Episode2 Pt4 $(E, r, 1, T)$

- MC (every-visit w. episode $C' \to D'' \to E' \to D' \to C \to D \to E \to T$):
- Update $G(s)$ backward:
  - $G(E) = R_T = 1, G(D) = R_{T-1} + \gamma G(E) = 0 + 1 = 1, G(C) = R_{T-2} + \gamma G(D) = 0 + 1 = 1$
  - $G(D') = R_{T-1} + \gamma G(C) = 0 + 1 = 1\ , G(E') = R_{T-1} + \gamma G(D') = 0 + 1 = 1$
  - $G(D'') = R_{T-1} + \gamma G(E') = 0 + 1 = 1, G(C') = R_{T-1} + \gamma G(D'') = 0 + 1 = 1$
- Update $V(s)$ forward:
  - $V(C) \leftarrow V(C) + \alpha[G(C') - V(C)] = .75 + .5(1 - .75) = 0.875$
  - $V(D) \leftarrow V(D) + \alpha[G(D'') - V(D)] = .75 + .5(1 - .75) = 0.875$
  - $V(E) \leftarrow V(E) + \alpha[G(E') - V(E)] = .75 + .5(1 - .75) = 0.875$
  - $V(D) \leftarrow V(D) + \alpha[G(D') - V(D)] = .875 + .5(1 - .875) \approx 0.938$
  - $V(C) \leftarrow V(C) + \alpha[G(C) - V(C)] = .875 + .5(1 - .875) \approx 0.938$
  - $V(D) \leftarrow V(D) + \alpha[G(D) - V(D)] = 0.938 + .5(1 - 0.938) = 0.969$
  - $V(E) \leftarrow V(E) + \alpha[G(E) - V(E)] = .875 + .5(1 - .875) \approx 0.938$

# TD vs. MC: Random Walk Performance

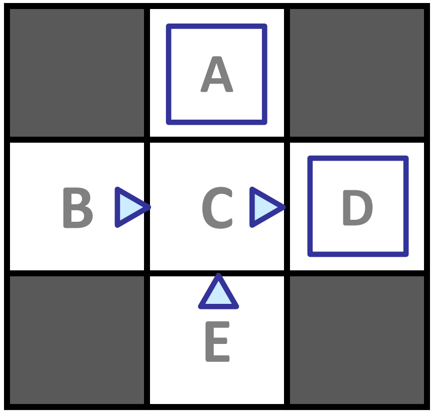
- Left fig shows values learned after various numbers of episodes on a single run of TD(0),
  - They are very close to the true values after 100 episodes, but they fluctuate indefinitely in response to the outcomes of the most recent episodes.
- Right fig shows Root Mean-Squared (RMS) error between the value function learned and the true value function, averaged over the five states, then averaged over 100 runs.
  - TD converges faster than MC. Higher learning rate $\alpha$ helps achieve faster convergence, but has large fluctuations.





# MC Prediction for MiniGW

## Input Policy π



*Assume:* γ = 1

## Observed Episodes (Training)

### Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

### Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

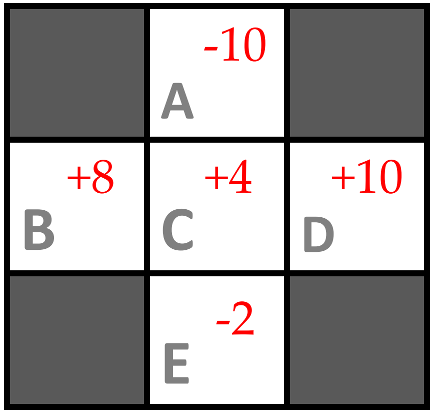
### Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

### Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

## Output Values

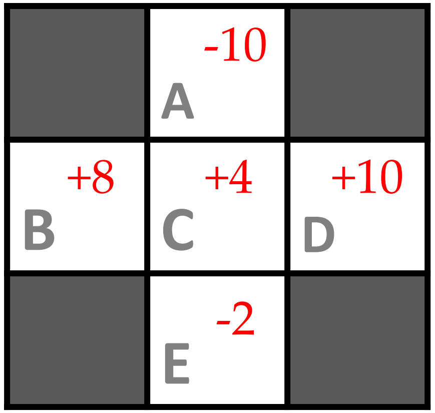


- State A:
  - Episode 4: $G_t(A) = -10$
  - $\hat{V}(A) = \frac{-10}{1} = -10,$
- State D:
  - Episodes 1,2, 4: $G_t(D) = 10$
  - $\hat{V}(D) = \frac{10 \cdot 3}{3} = 10,$
- State $B$:
  - Episodes 1 and 2: $G_t(B) = -1 - 1 + 10 = 8$; $\hat{V}(B) = \frac{1}{2}(8 + 8) = 8$
- State $C$:
  - Episodes 1,2,3: $G_t(C) = -1 + 10 = 9$
  - Episode 4: $G_t(C) = -1 - 10 = -11$
  - $\hat{V}(C) = \frac{1}{4}(9 + 9 + 9 - 11) = 4$
- State $E$:
  - Episode 3: $G_t(E) = -1 - 1 + 10 = 8$
  - Episode 4: $G_t(E) = -1 - 1 - 10 = -12$
  - $\hat{V}(E) = \frac{1}{2}(8 - 12) = -2$

# MC Prediction for MiniGW

- From Policy Evaluation, we have derived $V(B) = V(E)$. But the MC predicted value functions are inaccurate due to limited sampling:
  - $\hat{V}(B) = 8$, since both episodes 1 and 2 start from $B$ and end in $D$ with $V(D) = 10$;
  - $\hat{V}(E) = -2$, since episodes 3 and 4 start from $E$ and end in either $D$ or $A$ with $V(D) = 10, V(A) = -10$.
- If we sample more data, then we can estimate more accurate values.
- MC learning is not sample-efficient, since value function of each state must be learned separately.

Output Values



*If B and E both go to C under this policy, how can their values be different?*

# TD Learning, $\alpha = 0.5$

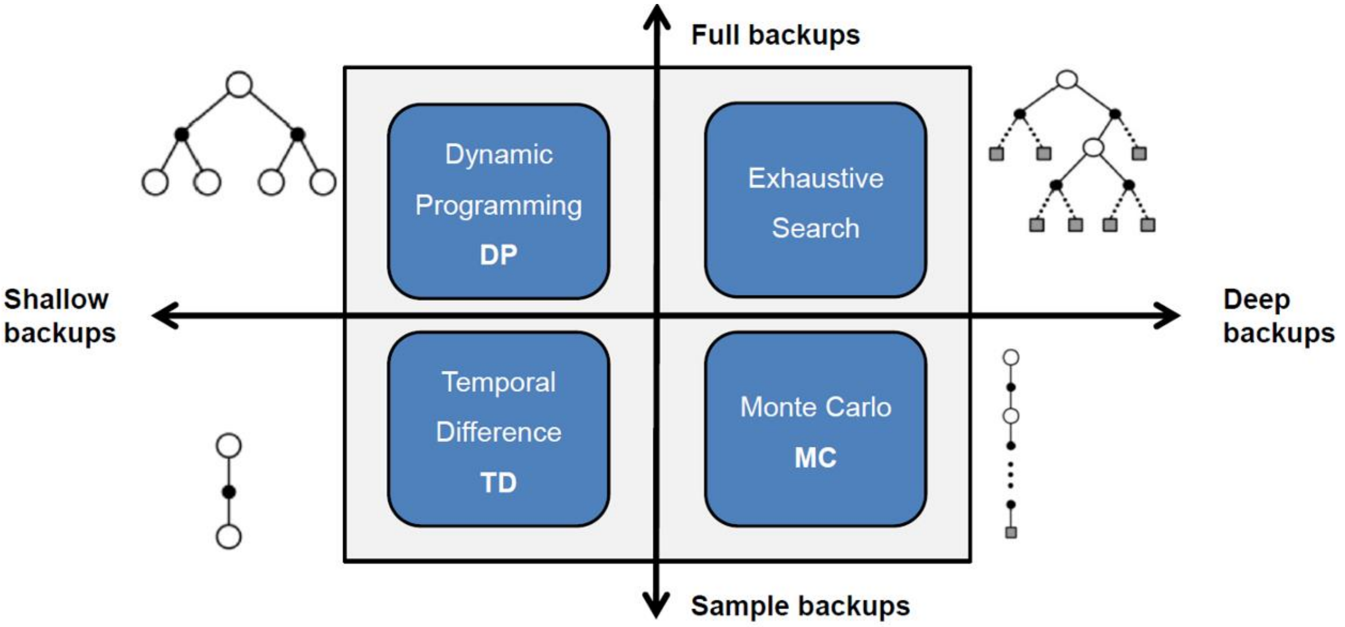
- 4 episodes with transitions {(B, east, C), (C, east, D)}.
- TD update: $v_\pi(B) \leftarrow_\alpha -1 + \gamma v_\pi\,(C); v_\pi(B) \leftarrow_\alpha -1 + \gamma v_\pi\,(D)$
- EP1:
- Transition (B, east, C): $V^\pi(B) \leftarrow 0 + \alpha(-1 + 0 - 0) = -.5$
- Transition (C, east, D): $V^\pi(C) \leftarrow 0 + \alpha(-1 + 10 - 0) = 4.5$
- EP2:
- Transition (B, east, C): $V^\pi(B) \leftarrow -.5 + \alpha(-1 + 4.5 - (-.5)) = 1.5$
- Transition (C, east, D): $V^\pi(C) \leftarrow 4.5 + \alpha(-1 + 10 - 4.5) = 6.75$
- EP3:
- Transition (B, east, C): $V^\pi(B) \leftarrow 1.5 + \alpha(-1 + 6.75 - 1.5) = 3.625$
- Transition (C, east, D): $V^\pi(C) \leftarrow 6.75 + \alpha(-1 + 10 - 6.75) = 7.875$
- EP4:
- Transition (B, east, C): $V^\pi(B) \leftarrow 3.625 + \alpha(-1 + 7.875 - 3.625) = 5.25$
- Transition (C, east, D): $V^\pi(C) \leftarrow 7.875 + \alpha(-1 + 10 - 7.875) = 8.4375$
- After many repetitions of the episode {(B, east, C), (C, east, D)}, $V^\pi(B) \approx 7, V^\pi(C) \approx 9$

# TD Learning, $\alpha = 0.9$

- 4 episodes with transitions {(B, east, C), (C, east, D)}.
- TD update: $v_\pi(B) \leftarrow_\alpha -1 + \gamma v_\pi\,(C); v_\pi(B) \leftarrow_\alpha -1 + \gamma v_\pi\,(D)$
- EP1:
- Transition (B, east, C): $V^\pi(B) \leftarrow 0 + \alpha(-1 + 0 - 0) = -.9$
- Transition (C, east, D): $V^\pi(C) \leftarrow 0 + \alpha(-1 + 10 - 0) = 8.1$
- EP2:
- Transition (B, east, C): $V^\pi(B) \leftarrow -.9 + \alpha(-1 + 8.1 - (-.9)) = 3.1$
- Transition (C, east, D): $V^\pi(C) \leftarrow 8.1 + \alpha(-1 + 10 - 8.1) = 8.91$
- EP3:
- Transition (B, east, C): $V^\pi(B) \leftarrow 3.1 + \alpha(-1 + 8.91 - 3.1) = 5.505$
- Transition (C, east, D): $V^\pi(C) \leftarrow 8.91 + \alpha(-1 + 10 - 8.91) = 8.991$
- EP4:
- Transition (B, east, C): $V^\pi(B) \leftarrow 5.505 + \alpha(-1 + 8.991 - 5.505) = 6.748$
- Transition (C, east, D): $V^\pi(C) \leftarrow 8.991 + \alpha(-1 + 10 - 8.991) = 8.9991$
- After many repetitions of the episode {(B, east, C), (C, east, D)}, $V^\pi(B) \approx 7, V^\pi(C) \approx 9$. Converges faster than $\alpha = 0.5$

Important

# Backup Diagrams: MC vs. TD vs. DP



Full backups

Dynamic Programming DP

Exhaustive Search

Shallow backups

Deep backups

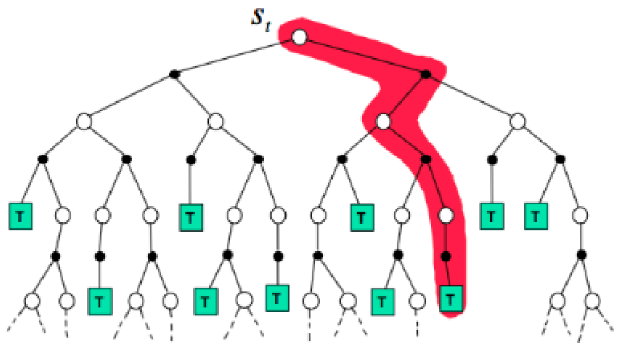
Temporal Difference TD

Monte Carlo MC

Sample backups

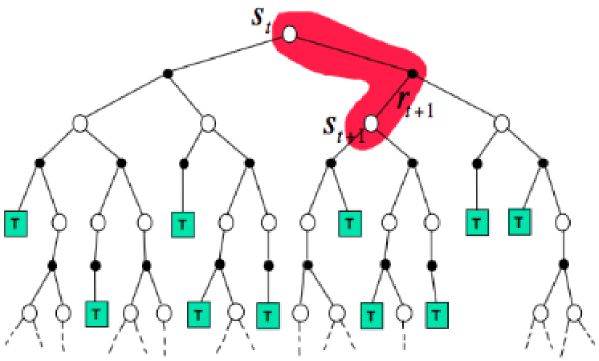
Monte-Carlo

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



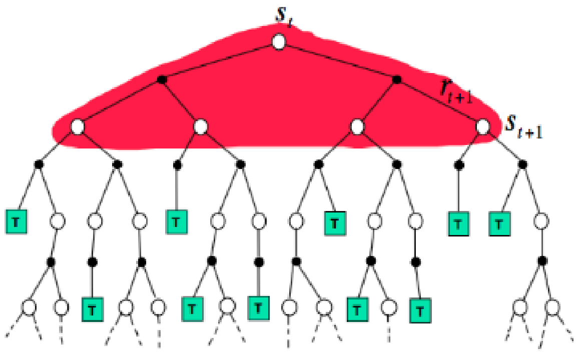
Temporal-Difference

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



Dynamic Programming

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



# MC vs. TD

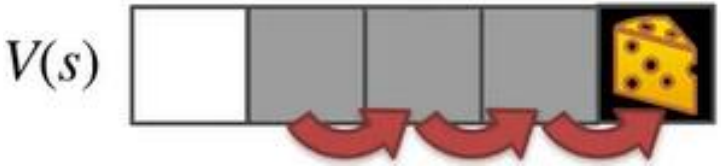
- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of each episode before return is known
- TD can learn without the final outcome
  - TD can learn from incomplete episodes
  - MC can only learn from complete episodes
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments
- MC has high variance, zero bias
  - Return $G_t \doteq \sum_{k=0}^{T-1} \gamma^k R_{t+k+1}$ is unbiased estimate of $v_\pi(S_t)$; converges to $v_\pi(S_t)$ (even with function approximation)
  - Return depends on many random actions, transitions, rewards in each episode
  - Not very sensitive to initial value
- TD has low variance, some bias
  - TD target $R_{t+1} + \gamma V(S_{t+1})$ is biased estimate of $v_\pi(S_t)$; TD(0) converges to $v_\pi(S_t)$ (but not always with function approximation)
  - TD target depends on one random action, transition, reward
  - More sensitive to initial value
- MC does not exploit Markov property
  - More effective in non-Markov environments, e.g., Partially Observed MDP (POMDP)
- TD exploits Markov property
  - Does not work well for POMPDP

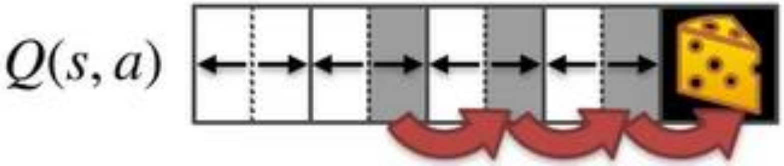
# Outline

- Monte Carlo Methods
- TD-Learning
- Sarsa & Q-Learning
- Function Approximation

# Problems with TD Learning

- TD is a model-free way to learn $V(S)$ by sampling
- However, if we want to get the optimal policy, we need the MDP model $p(r, s'|s, a)$ to go from $V(S)$ to $Q(S, A)$
  - $q(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma v(s')]$
- Then use greedy action selection
  - $\pi(s) = \underset{a}{\operatorname{argmax}} q(s, a)$
- Sarsa and Q learning: learn $Q(S, A)$ instead of $V(S)$, for use in Generalized Policy Iteration (GPI) for control. This makes action selection model-free too.





Important

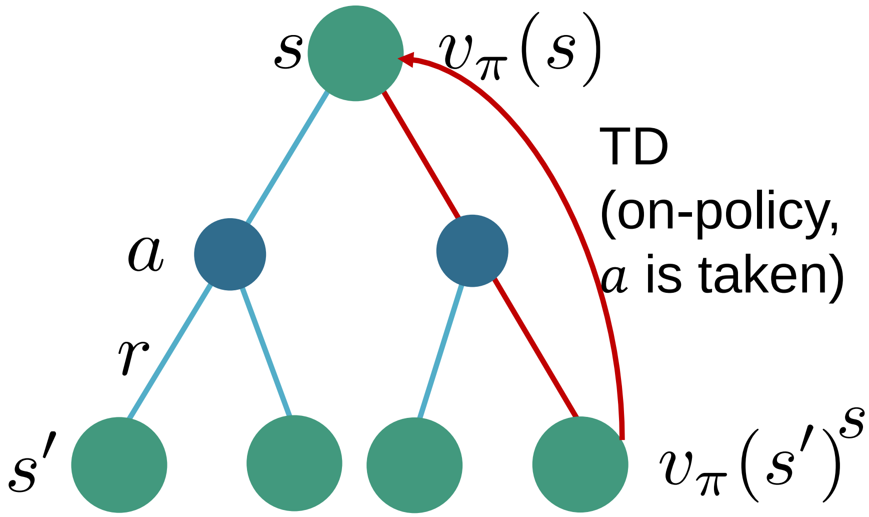
# TD, Sarsa, Q Learning

- TD solves [BEV] by sampling:
  - $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
- Sarsa and Expected Sarsa solve [BEA] by sampling:
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t))$
- Q Learning solves [BOA] by sampling:
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t))$

- [BEV] Bellman Expectation Equation for State Value Function:
  - $v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_\pi(s')]$
- [BEA] Bellman Expectation Equation for Action Value Function
  - $q_\pi(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')]$
- [BOA] Bellman Optimality Equation for Optimal Action Value Function:
  - $q_*(s, a) = \sum_{r,s'} p(r, s'|s, a) \left[r + \gamma \max_{a'} q_*(s', a')\right]$

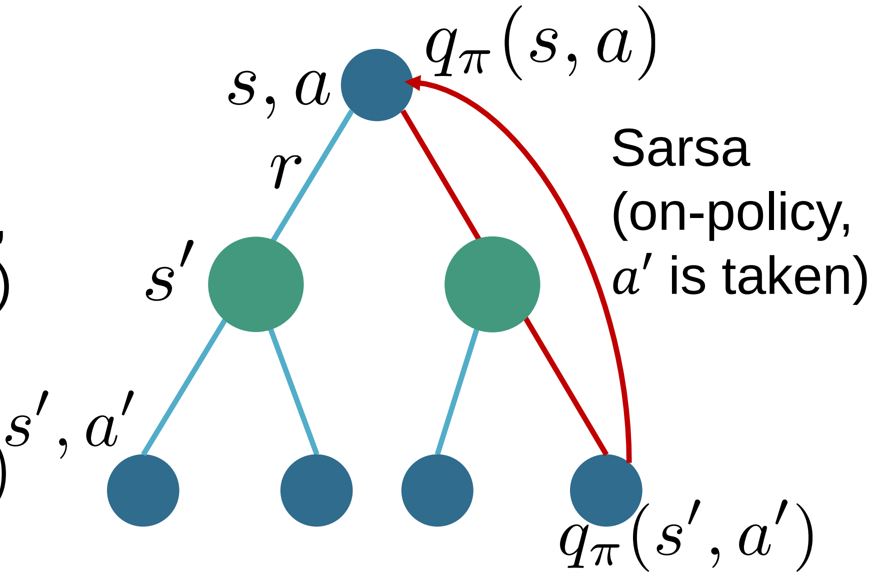
Important

# Backup Diagrams



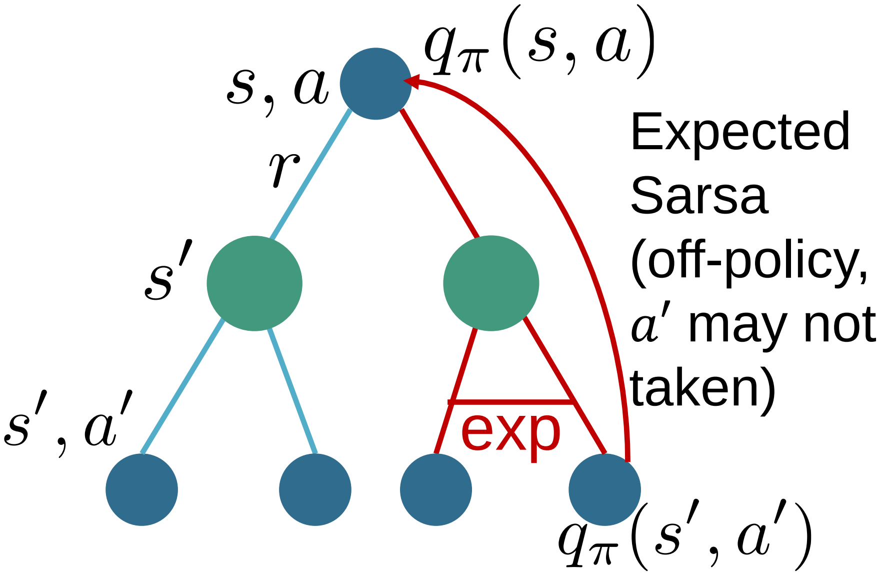
TD (on-policy, $a$ is taken)

Bellman Exp Eqn for $v_\pi(s)$



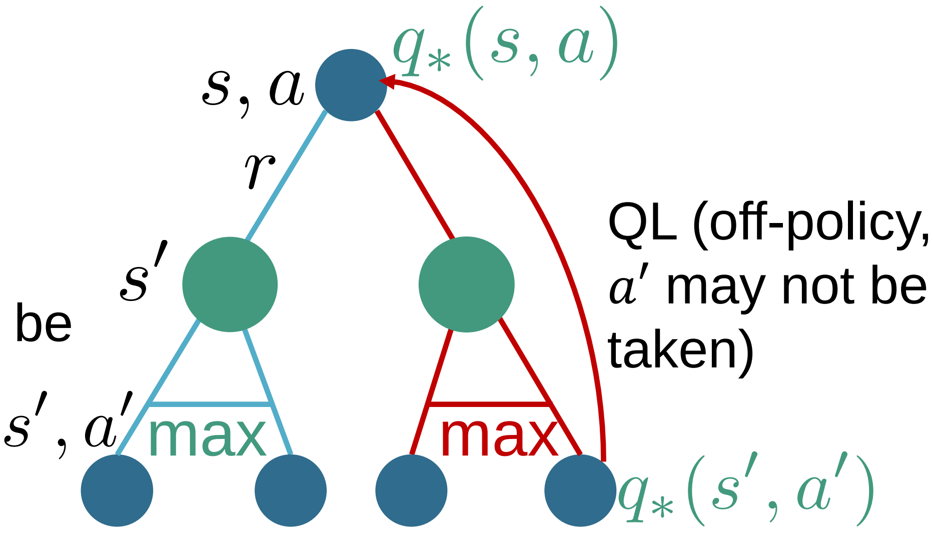
Sarsa (on-policy, $a'$ is taken)

Bellman Exp Eqn for $q_\pi(s, a)$



Expected Sarsa (off-policy, $a'$ may not be taken)

Bellman Exp Eqn for $q_\pi(s, a)$

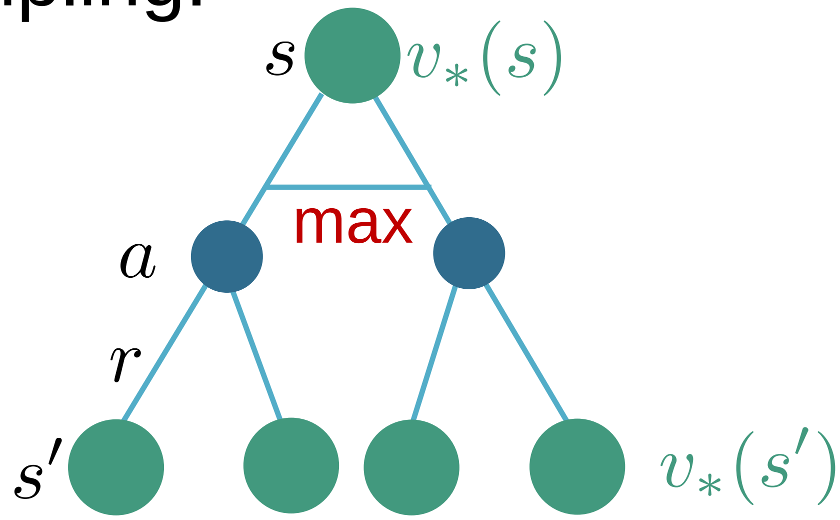


QL (off-policy, $a'$ may not be taken)

Bellman Opt Eqn for $q_*(s, a)$

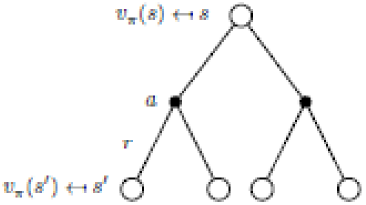
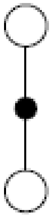
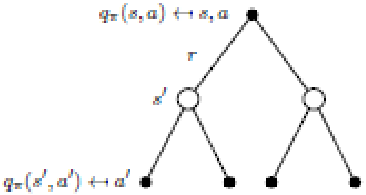
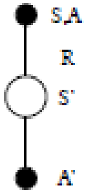
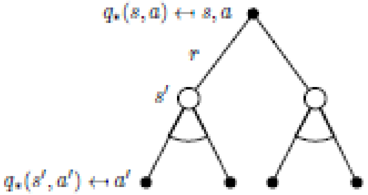
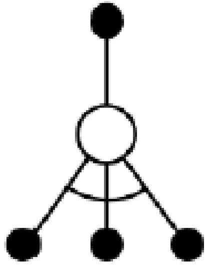
# Did We Miss One Bellman Equation?

- [BEV] Bellman Optimality Equation for Optimal State Value Function:
  - $v_*(s) = \max_a \sum_{r,s'} p(r, s'|s, a) \left[r + \gamma v_*(s')\right]$
- Due to $\max_a$ in front, it is not an expectation over a distribution, hence cannot solve for $v_*(s)$ by sampling.

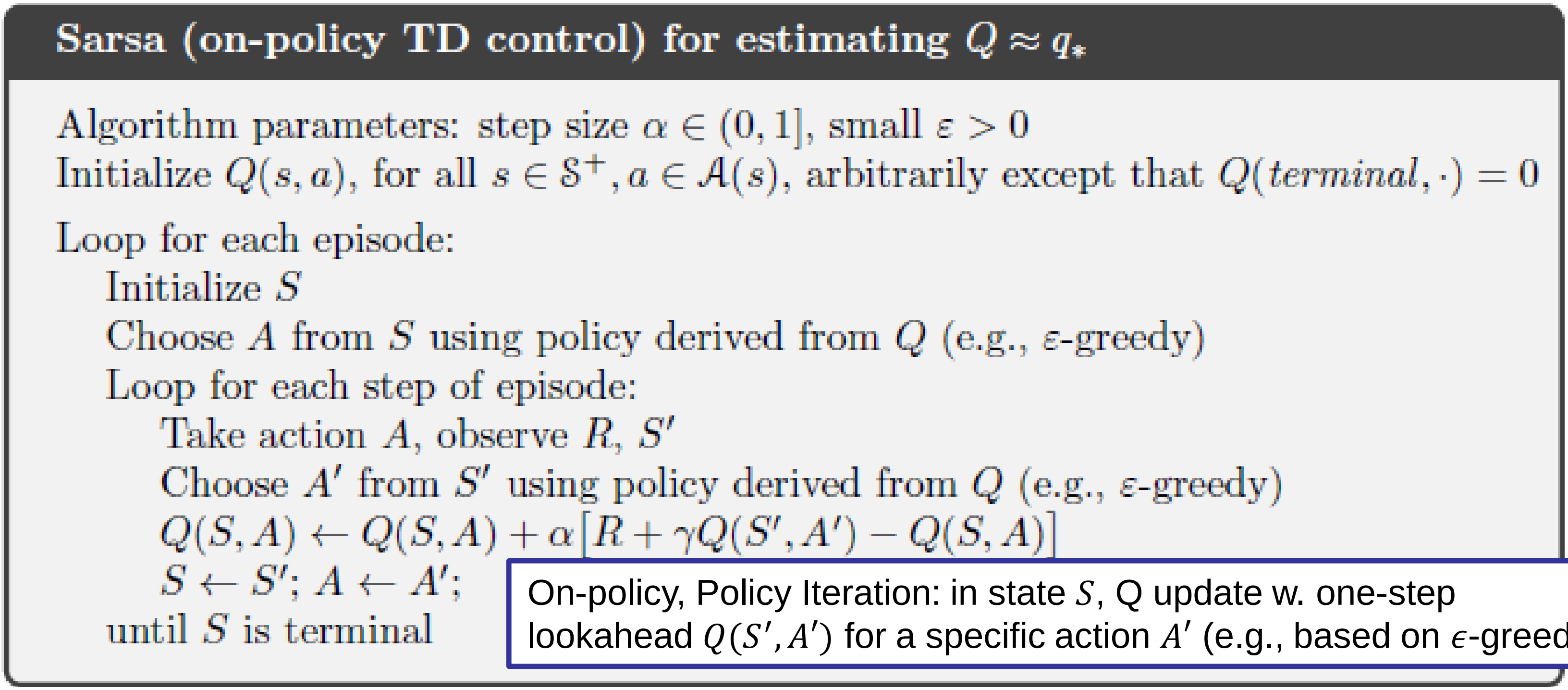


Bellman Opt Eqn for $v_*(s)$

# Another View

| | *Full Backup (DP)* | *Sample Backup (TD)* |
|---|---|---|
| Bellman Expectation Equation for $v_\pi(s)$ |   $v_\pi(s) \leftrightarrow s$<br>$a$<br>$r$<br>$v_\pi(s') \leftrightarrow s'$<br>Iterative Policy Evaluation | <br>TD Learning |
| Bellman Expectation Equation for $q_\pi(s, a)$ |   $q_\pi(s,a) \leftrightarrow s,a$<br>$r$<br>$s'$<br>$q_\pi(s',a') \leftrightarrow a'$<br>Q-Policy Iteration |  <br>Sarsa |
| Bellman Optimality Equation for $q_*(s, a)$ |   $q_*(s,a) \leftrightarrow s,a$<br>$r$<br>$s'$<br>$q_*(s',a') \leftrightarrow a'$<br>Q-Value Iteration | <br>Q-Learning |

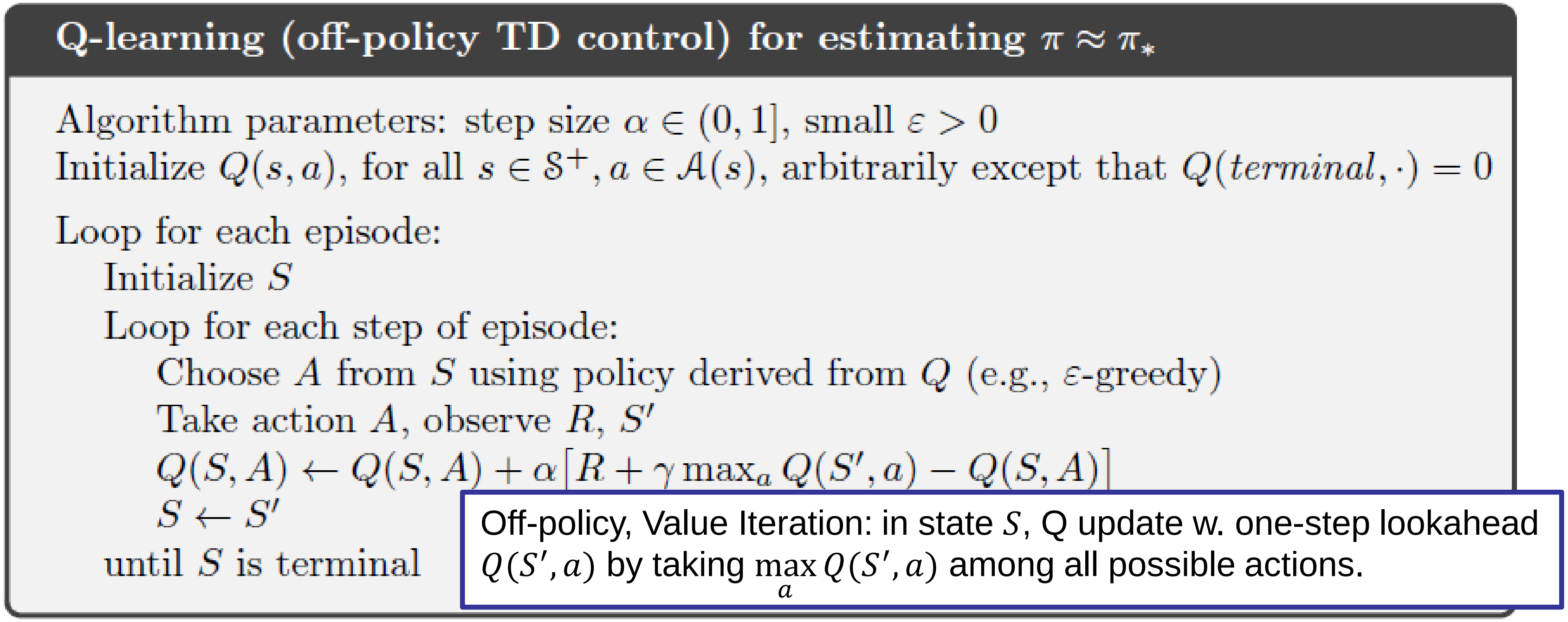
From https://www.davidsilver.uk/teaching/. Expected Sarsa is missing here.



**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 $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-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
```

On-policy, Policy Iteration: in state $S$, Q update w. one-step lookahead $Q(S',A')$ for a specific action $A'$ (e.g., based on $\epsilon$-greedy).



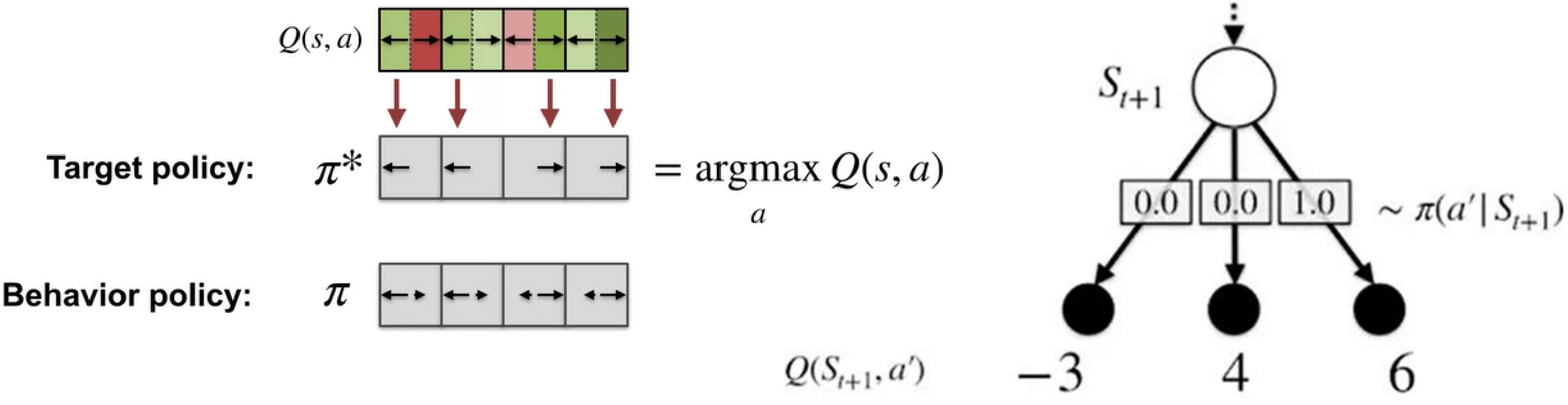
**Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$**

```
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., $\varepsilon$-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
```

Off-policy, Value Iteration: in state $S$, Q update w. one-step lookahead $Q(S',a)$ by taking $\max_a Q(S',a)$ among all possible actions.

# QL is Off-Policy

- QL's target policy $\pi$ is always greedy w.r.t $Q(s,a)$
  - $\sum_{a'} \pi(a'|S_{t+1})\, Q(S_{t+1}, a') = \max_{a'} Q(S_{t+1}, a')$
- Behavior policy b is typically more exploratory, e.g., $\epsilon$-greedy, or uniform random, or even arbitrary policy.
- No need for Importance Sampling.

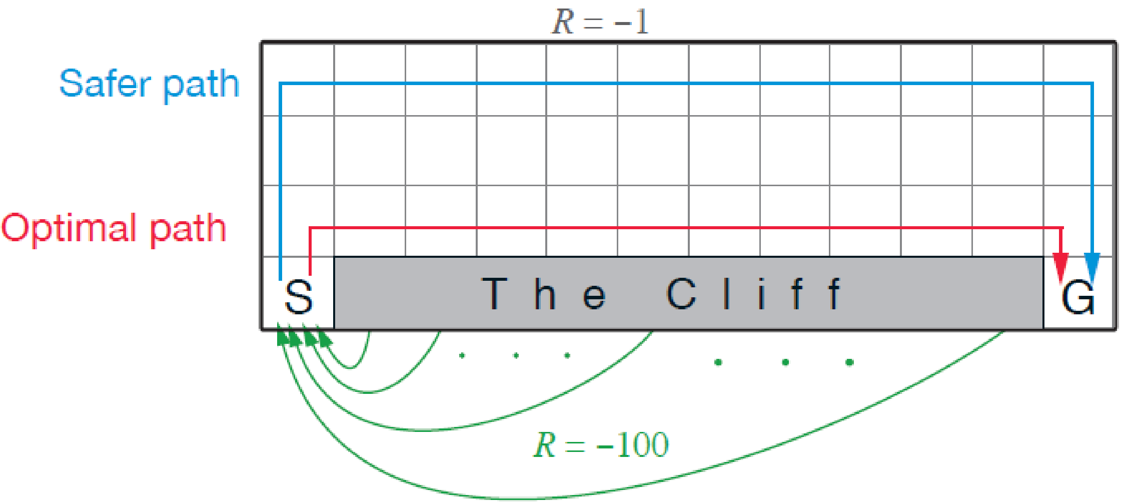


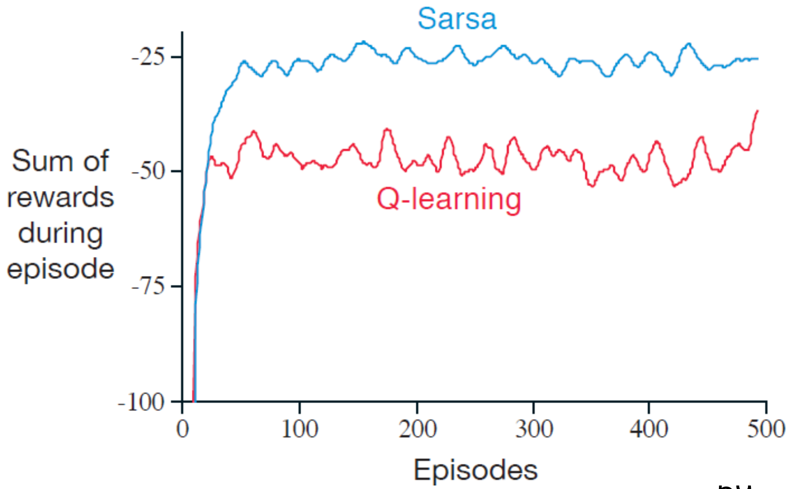
# Expected Sarsa and QL

- Expected Sarsa and QL are both off-policy.
  - Target for Expected Sarsa: $R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1})\, Q(S_{t+1}, a')$
  - Target for QL: $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$
- Expected Sarsa improves over Sarsa by eliminating variance due to the random selection of $A_{t+1}$, with cost of increased computation overhead.

# Example 6.6: Cliff Walking

- Reward is −1 on all transitions except those into the region marked “The Cliff.” Stepping into this region incurs a reward of −100 and sends the agent instantly back to the start. Env is deterministic.
- Graph shows the performance of the Sarsa and Q-learning methods with $\epsilon$-greedy action selection, $\epsilon = 0.1$. After an initial transient, Q-learning learns values for the optimal policy, that which travels right along the edge of the cliff. Unfortunately, this results in its occasionally falling off the cliff because of the $\epsilon$- greedy action selection. Sarsa, on the other hand, takes the action selection into account and learns the longer but safer path through the upper part of he grid. Although Q-learning learns the values of the optimal policy, its online performance is worse than that of Sarsa.
- If $\epsilon$ were gradually reduced to $0$, then both methods would asymptotically converge to the optimal policy.

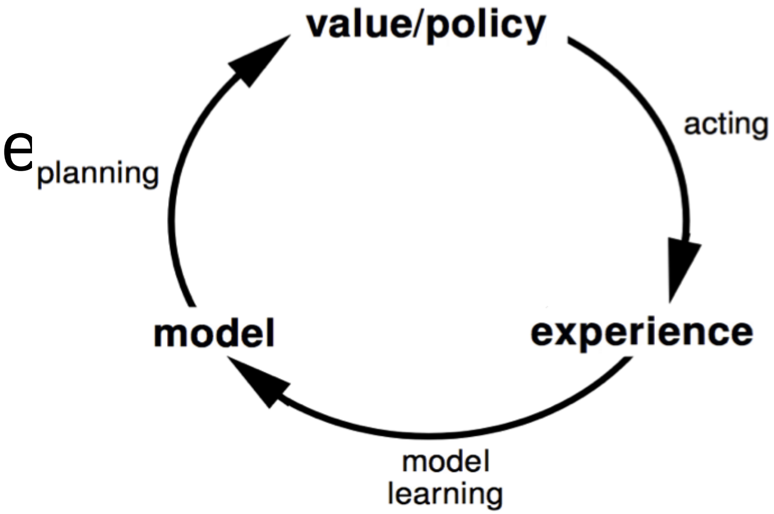


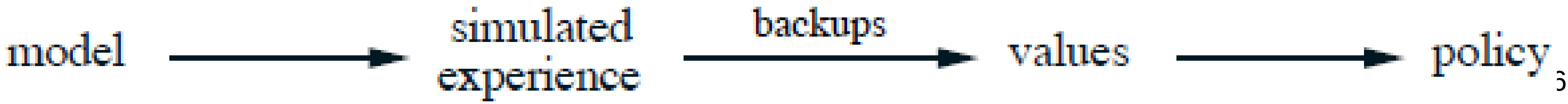


Recall

# Model-Based RL

- If MDP is not available, we can use Model-Based RL:
- Step 1: Learn empirical MDP model
  - Execute some policy $\pi$ (may be random), and keeping track of outcomes $r, s'$ for each $s, a$ in the observed episodes. These form the training set for supervised learning of the model $p(r, s'|s, a)$.
- Step 2: Do planning w. the learned MDP w. Dynamic Programming (Value Iteration or Policy Iteration)



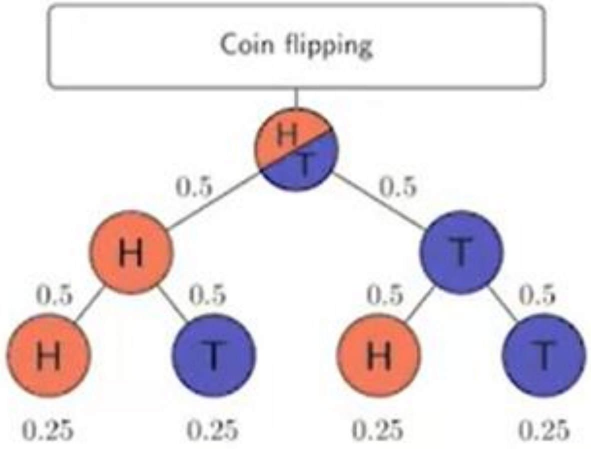


# Two Types of Models

- Sample models: produce just one of the possible outcomes, sampled according to the probabilities.
  - Can approximate probability of each outcome; requires little memory.
- Distribution models: produce the complete probability distribution of all possible outcomes.
  - Can compute exact probability of each expected outcome; requires more memory for storage.
  - Ex. Table Lookup, Linear Expectation, Linear Gaussian, Gaussian Process, Deep Belief Network…
- Dynamic Programming approaches (Value Iteration and Policy Iteration) require distribution models for state-space planning. Q-planning can use either type of model.

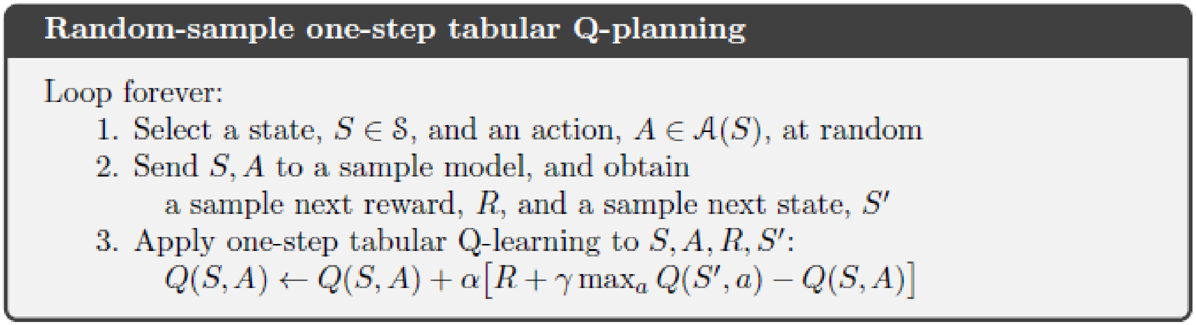
**Sample**

**Distribution**



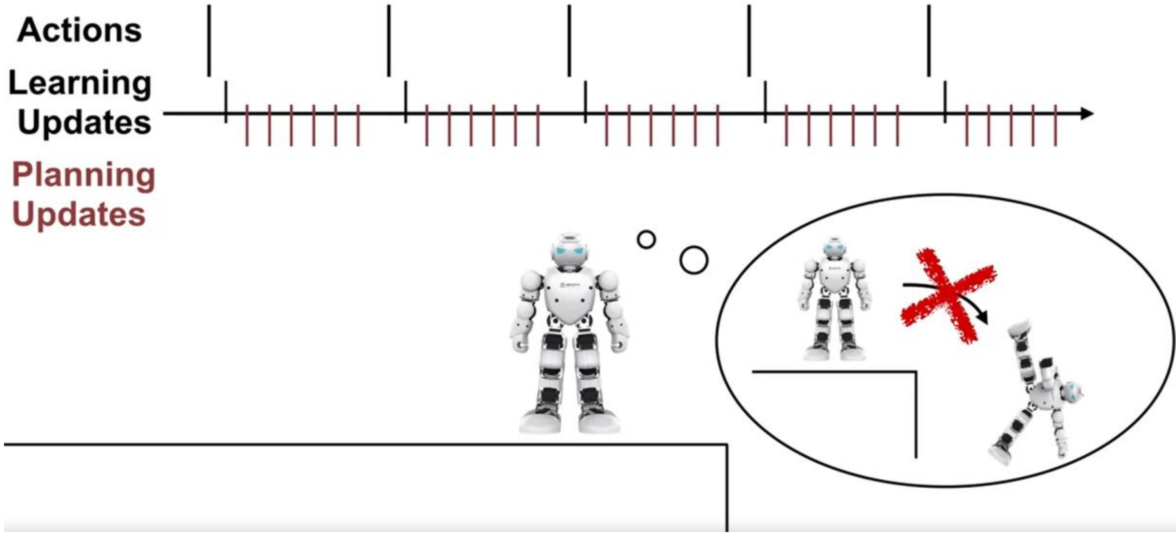
# Random-sample one-step tabular Q-planning

- Planning uses simulated/imagined experience generated by a model.
- Learning uses real experience generated by the environment.
- They can be combined (in Dyna-Q): an agent uses planning to learn that moving right and falling off the cliff is bad, so it avoids the moving right action without actually doing it.



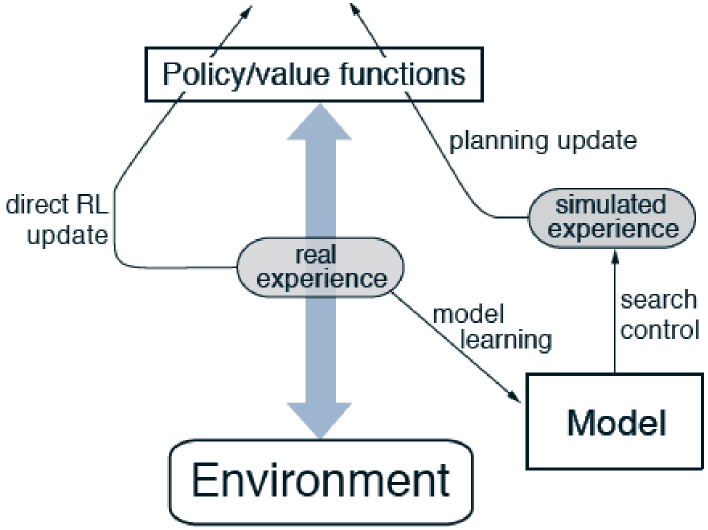
**Random-sample one-step tabular Q-planning**

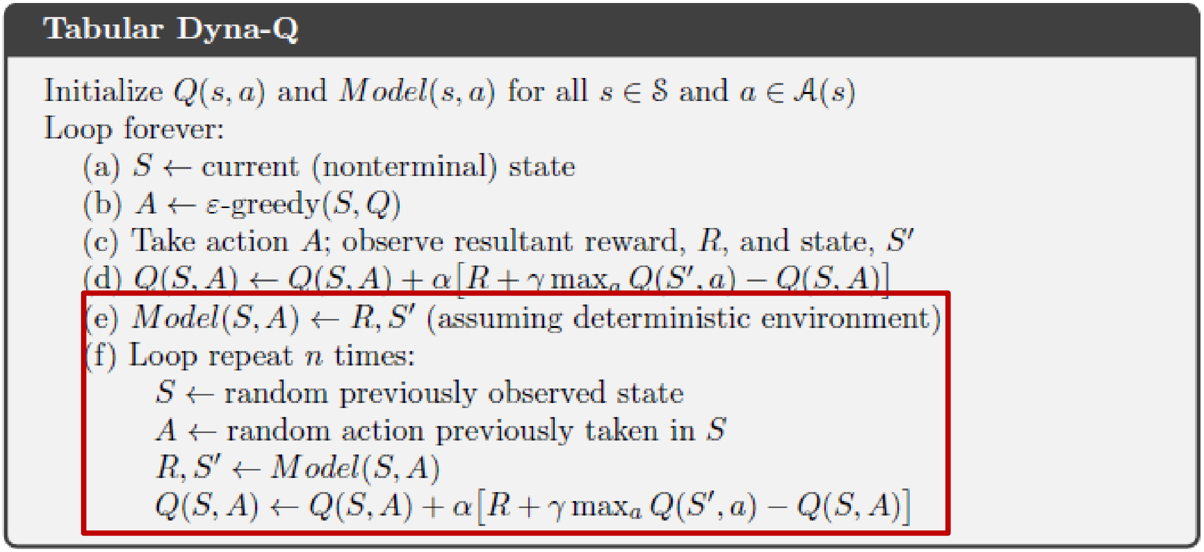
```
Loop forever:
    1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
    2. Send $S, A$ to a sample model, and obtain
           a sample next reward, $R$, and a sample next state, $S'$
    3. Apply one-step tabular Q-learning to $S, A, R, S'$:
           $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
```



# Dyna-Q: Integrated Planning, Acting, and Learning

- Indirect RL (model-learning then Q planning): real experience is used to improve the model (to make it more accurately match the real environment), which is used for planning.
- Direct RL w Q learning: real experience is used to directly improve the value function and policy.
- Tabular Dyna-Q = Q learning + Q planning
  - Model-learning is table-based and assumes deterministic environment. After each transition $S_t, A_t \rightarrow R_{t+1}, S_{t+1}$, the model records in its table entry for $S_t, A_t$ the prediction that $R_{t+1}, S_{t+1}$ will deterministically follow. If the model is queried with a state–action pair that has been experienced before, it simply returns the last-observed next state and next reward as its prediction.
  - During planning, the Q-planning algorithm randomly samples from state–action pairs that have previously been experienced.





```
Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S, Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    (f) Loop repeat $n$ times:
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random action previously taken in $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
```

# Dyna-Q Maze Example

- 1st episode: agent follows random policy and stumbles upon the goal G. Afterwards, policy of only one state leading to G is updated.
  - QL ($n = 0$): all experience before reaching G is discarded.
  - Dyna-Q ($n > 0$): learn a model by keeping track of all experience before reaching G: makes better use of env interactions.
- Future episodes:
  - QL ($n = 0$): each episode adds only one additional step to the policy (influences the neighboring states)
  - Dyna ($n > 0$): agent uses the learned model to plan better policies for all previously-visited states. By the end of the 3rd episode a complete optimal policy has been found (right fig.).

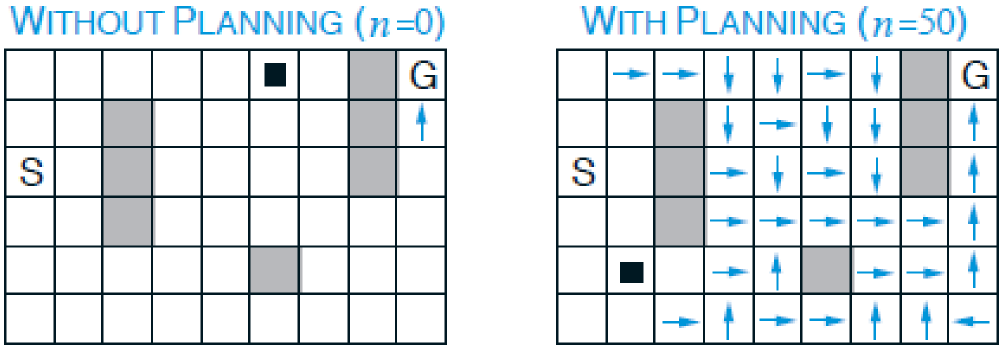
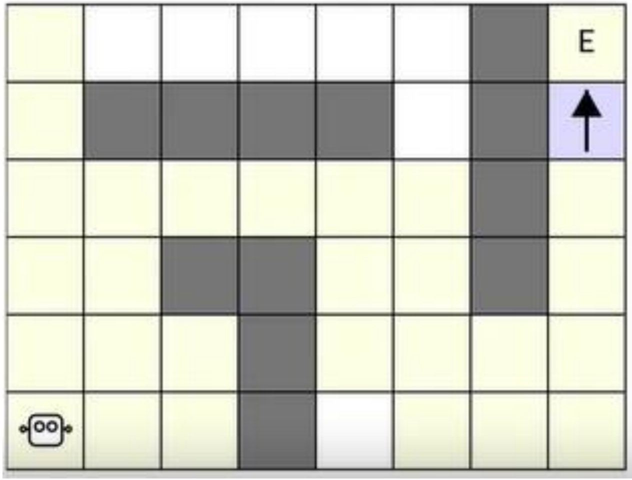


**Figure 8.3:** Policies found by planning and nonplanning Dyna-Q agents halfway through the second episode. The arrows indicate the greedy action in each state; if no arrow is shown for a state, then all of its action values were equal. The black square indicates the location of the agent. ■

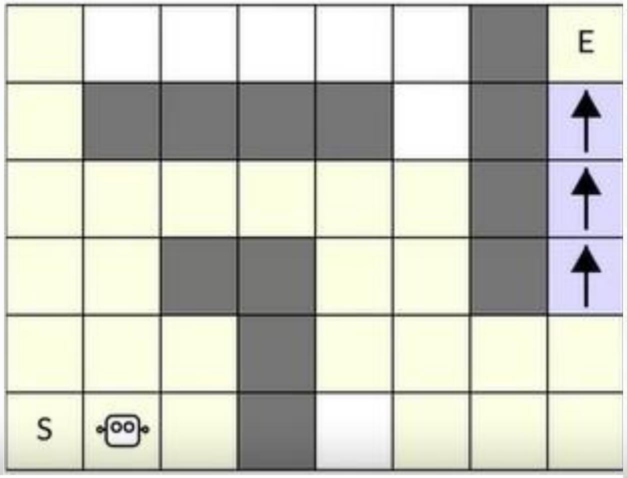
# Dyna-Q Maze Example ($n = 100$)

Number of actions taken: 184



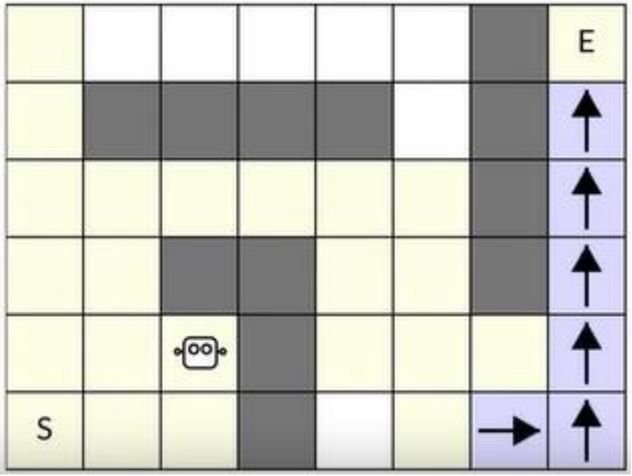
Number of steps planned: 100

Number of actions taken: 185



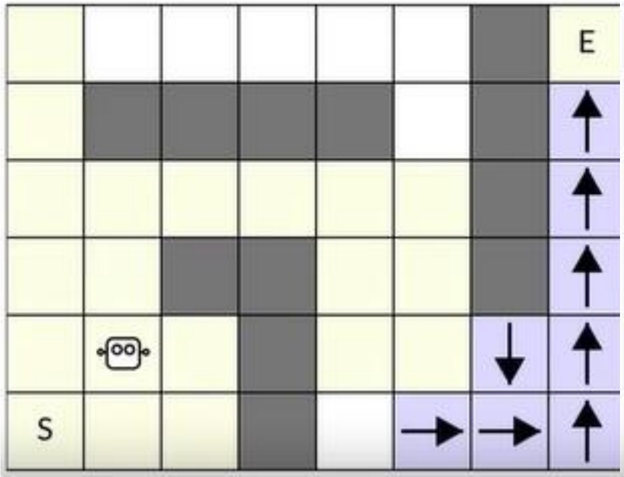
Number of steps planned: 300

Number of actions taken: 187



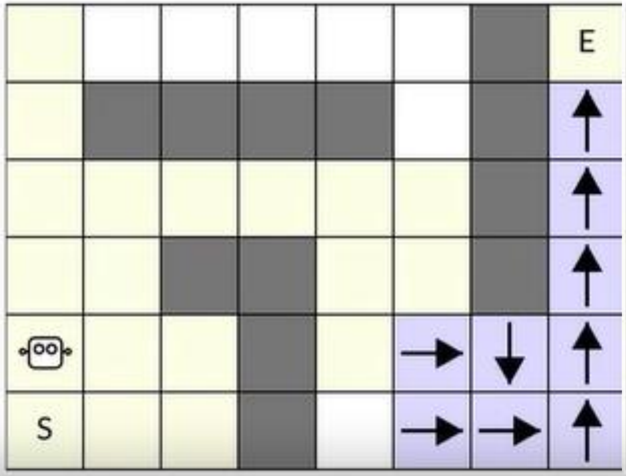
Number of steps planned: 500

Number of actions taken: 189



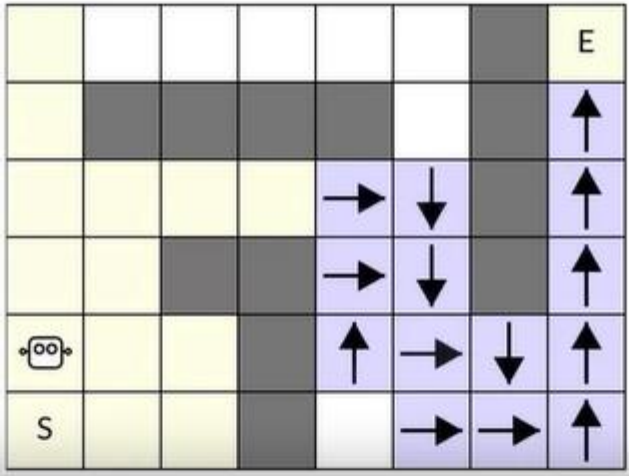
Number of steps planned: 600

Number of actions taken: 190



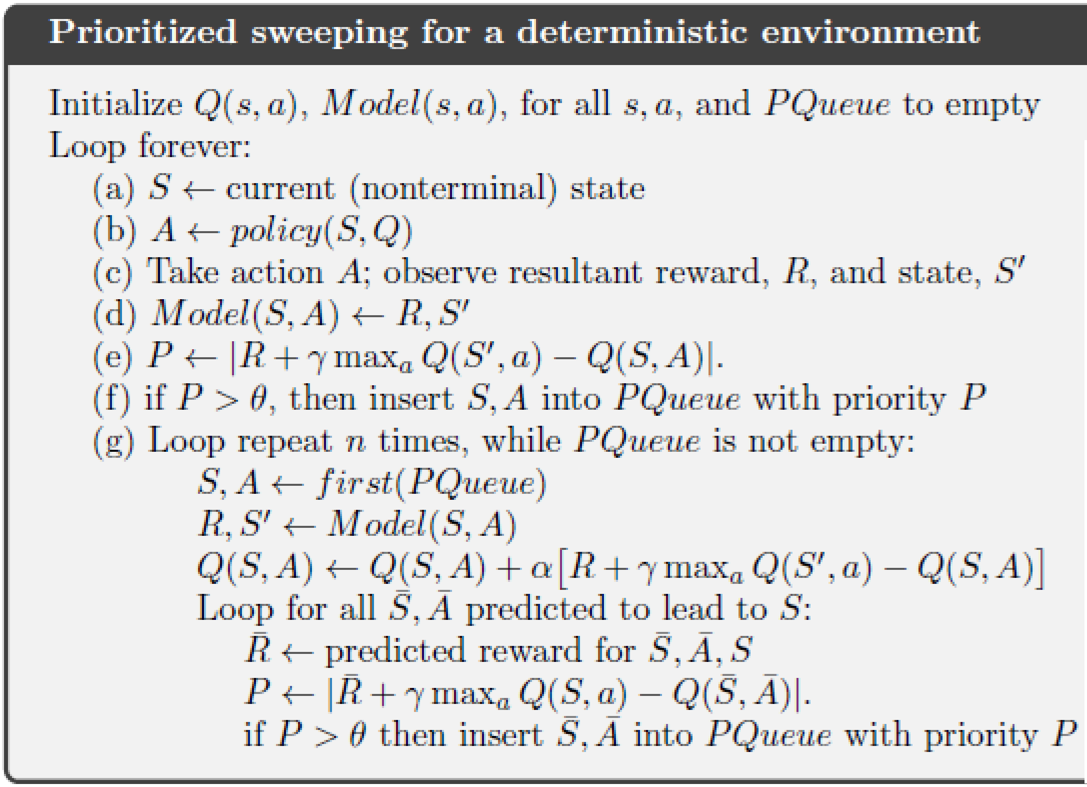
Number of steps planned: 1100

Number of actions taken: 195



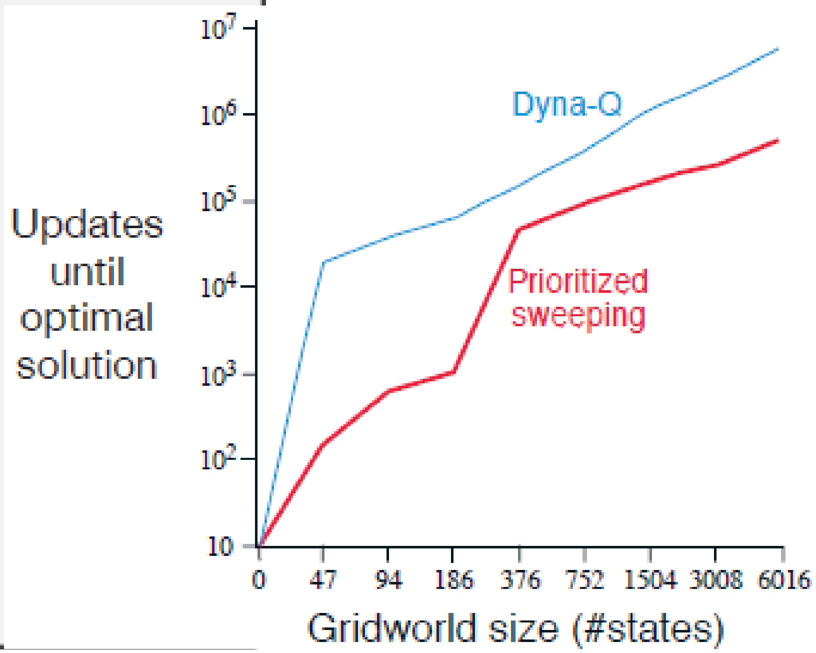
# Prioritized Sweeping

- Prioritized sweeping focuses backward on the predecessors of states whose values have recently changed
- A queue is maintained of every state–action pair whose estimated value would change nontrivially if updated , prioritized by the size of the change. When the top pair in the queue is updated, the effect (TD error) on each of its predecessor pairs is computed. If the effect is greater than some small threshold, then the pair is inserted in the queue with the new priority In this way the effects of changes are efficiently propagated backward until quiescence.
- e.g., for Maze example, work backward from the goal state by giving higher priority to states leading to the goal state than those far away from it.



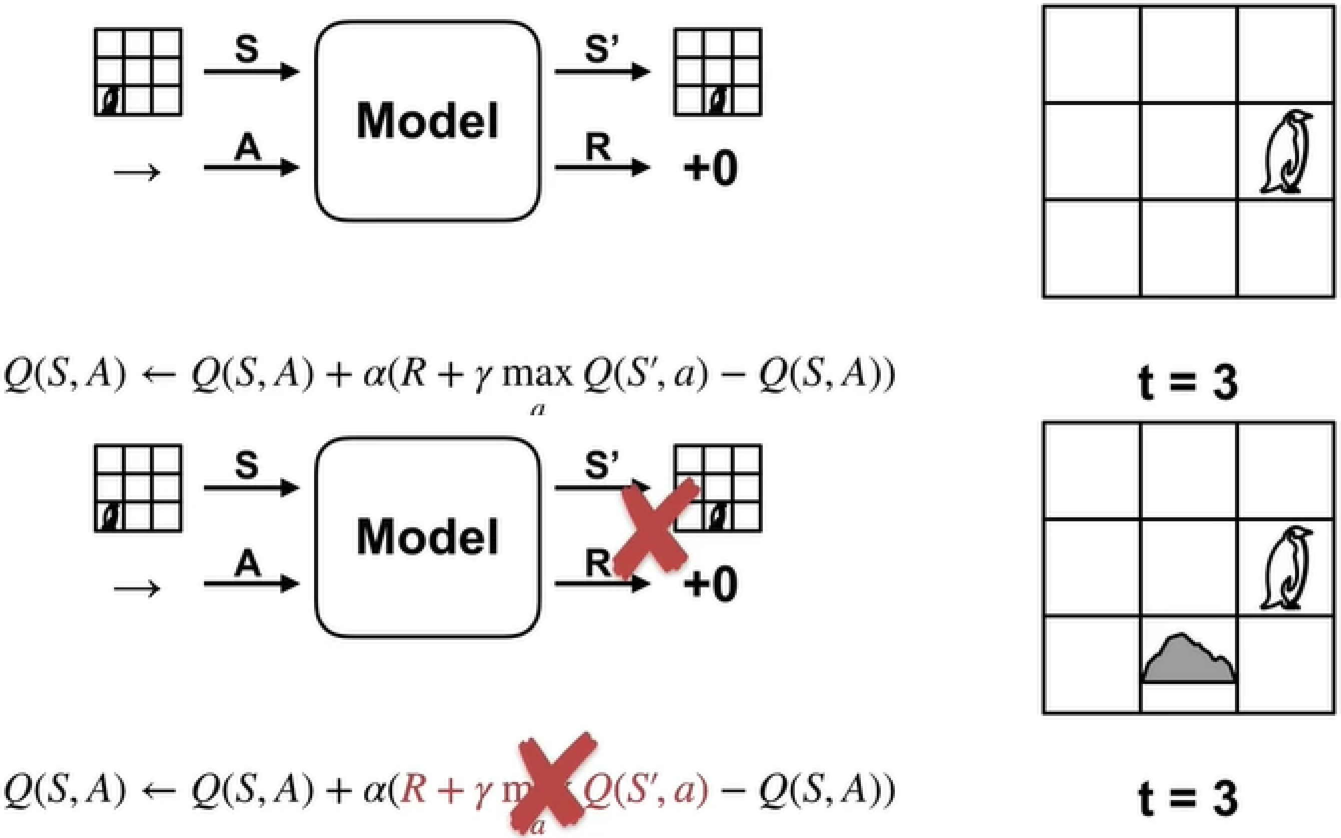
**Prioritized sweeping for a deterministic environment**

```
Initialize $Q(s,a)$, $Model(s,a)$, for all $s, a$, and $PQueue$ to empty
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow policy(S, Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Model(S, A) \leftarrow R, S'$
    (e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$.
    (f) if $P > \theta$, then insert $S, A$ into $PQueue$ with priority $P$
    (g) Loop repeat $n$ times, while $PQueue$ is not empty:
            $S, A \leftarrow first(PQueue)$
            $R, S' \leftarrow Model(S, A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
            Loop for all $\bar{S}, \bar{A}$ predicted to lead to $S$:
                $\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
                $P \leftarrow |\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})|$.
                if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority $P$
```



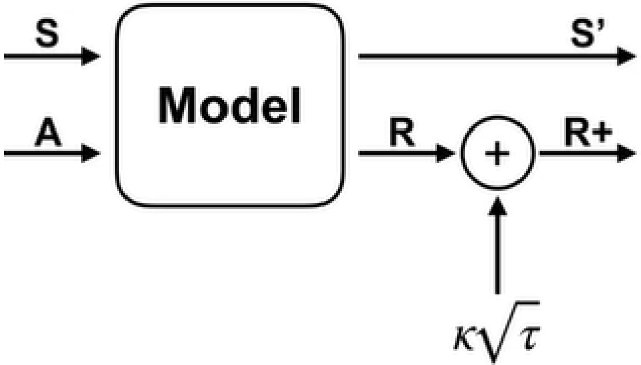
# Planning w. Inaccurate Model

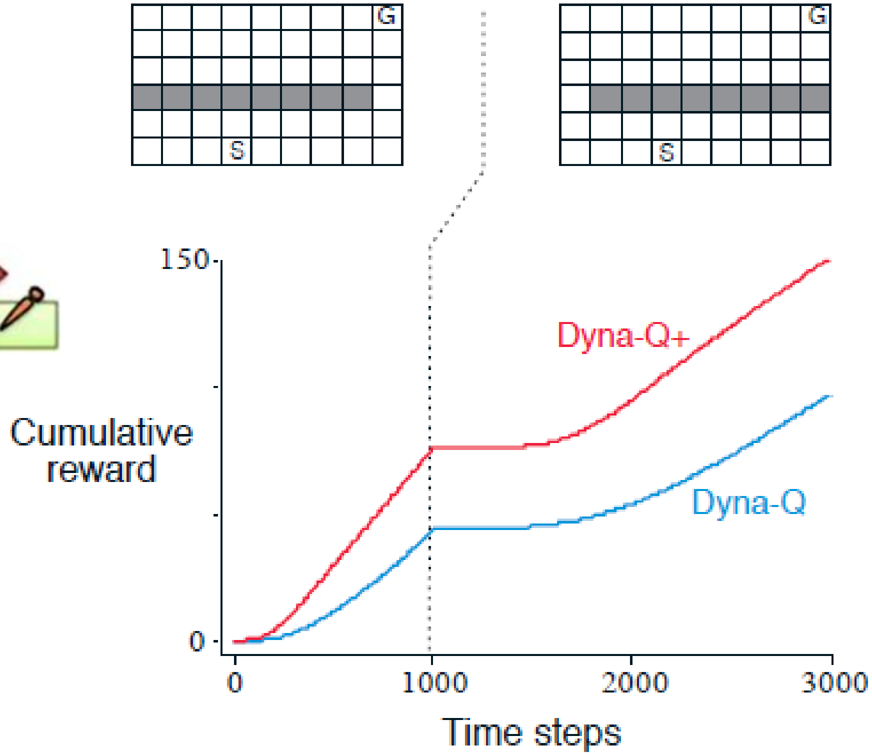
- Models may be incorrect because the environment is stochastic and only a limited number of samples have been observed, or because the model was learned using function approximation that has generalized imperfectly, or simply because the environment has changed and its new behavior has not yet been observed.
- When the model is incorrect, the planning process is likely to compute a suboptimal policy. But the error will be later discovered and corrected by real experience by exploration.


S
A
→
Model
S'
R
+0
$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_a Q(S',a) - Q(S,A))$
t = 3
S
A
→
Model
S'
R
+0
$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_a Q(S',a) - Q(S,A))$
t = 3

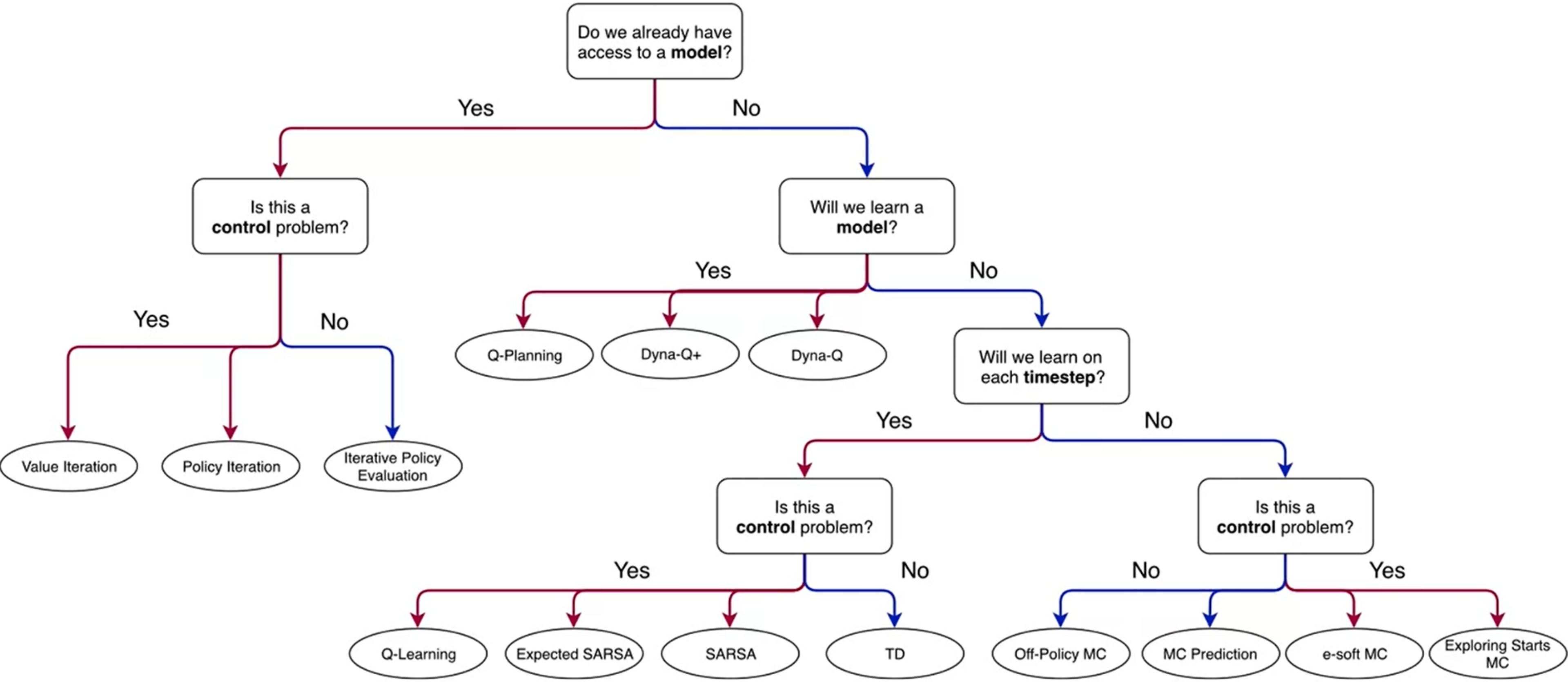
# Dyna-Q+

- Exploration (explore env to improve model accuracy) vs exploitation (use current model to improve policy).
- Dyna-Q+ is Dyna-Q with an exploration bonus that encourages exploration.
  - If the modeled reward for a transition is $r$, and the transition has not been tried in $\tau$ time steps, then planning updates are done as if that transition produced a reward of $r + \kappa\sqrt{\tau}$, for some small $\kappa$.
- Right fig: The left environment was used for the first 1000 steps, the right environment for the rest.





# Tabular Methods Summary

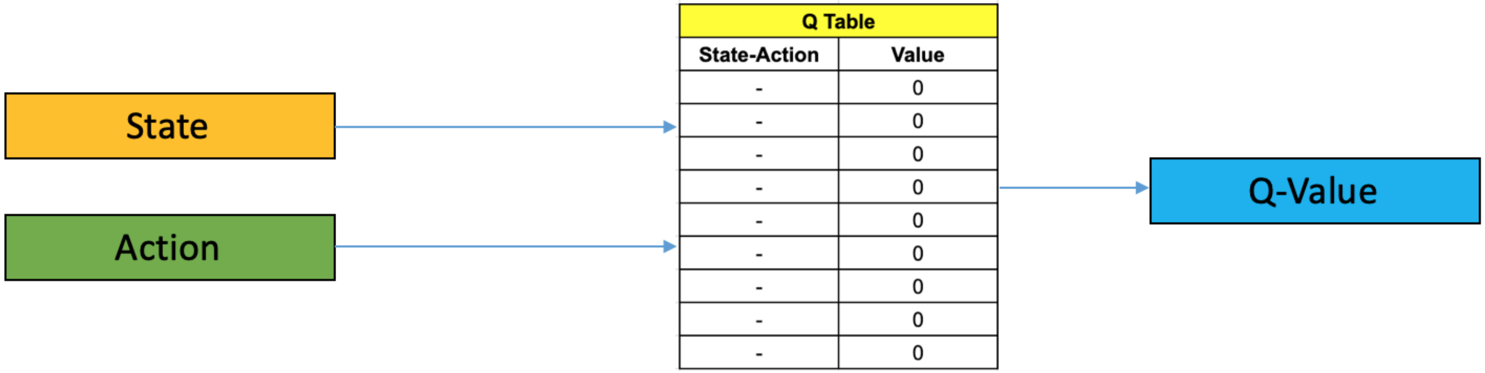


Do we already have access to a **model**?
Yes
No
Is this a **control** problem?
Will we learn a **model**?
Yes
No
Yes
No
Q-Planning
Dyna-Q+
Dyna-Q
Will we learn on each **timestep**?
Value Iteration
Policy Iteration
Iterative Policy Evaluation
Yes
No
Is this a **control** problem?
Is this a **control** problem?
Yes
No
No
Yes
Q-Learning
Expected SARSA
SARSA
TD
Off-Policy MC
MC Prediction
e-soft MC
Exploring Starts MC

# Outline

- Monte Carlo Methods
- TD-Learning
- Sarsa & Q-Learning
- Function Approximation

# Q Learning vs. DQN



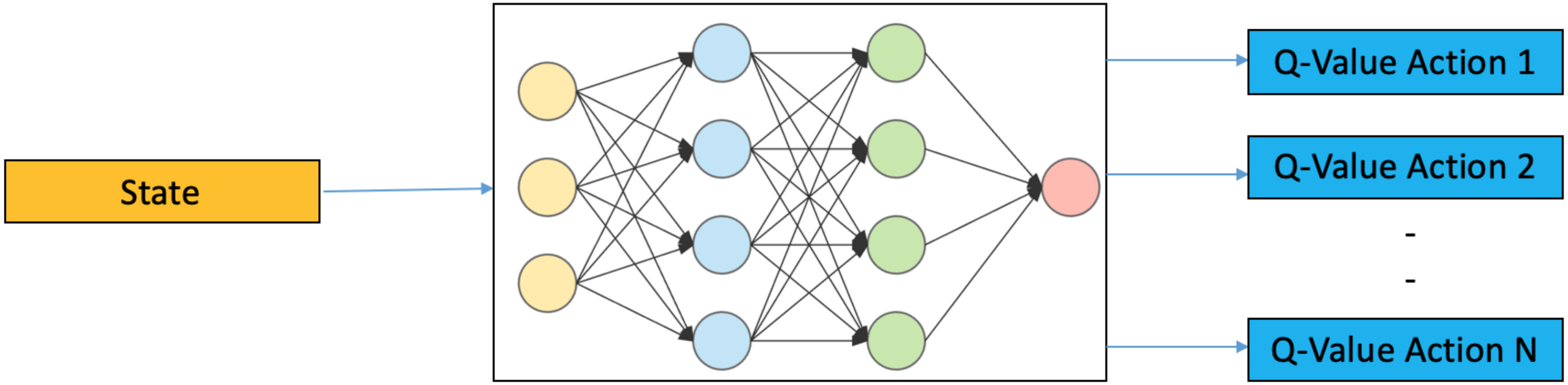
State

Action

| Q Table | |
|---|---|
| State-Action | Value |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |
| - | 0 |

Q-Value

Q Learning



State

Q-Value Action 1

Q-Value Action 2

-

-

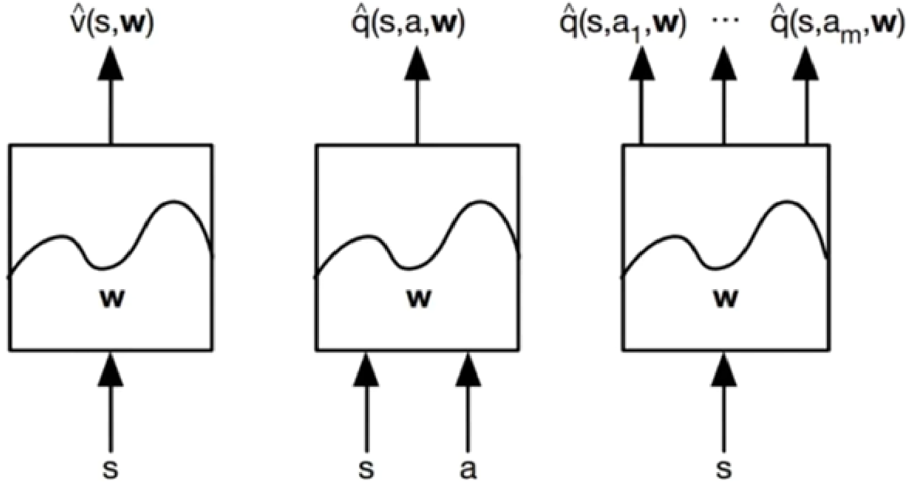
Q-Value Action N

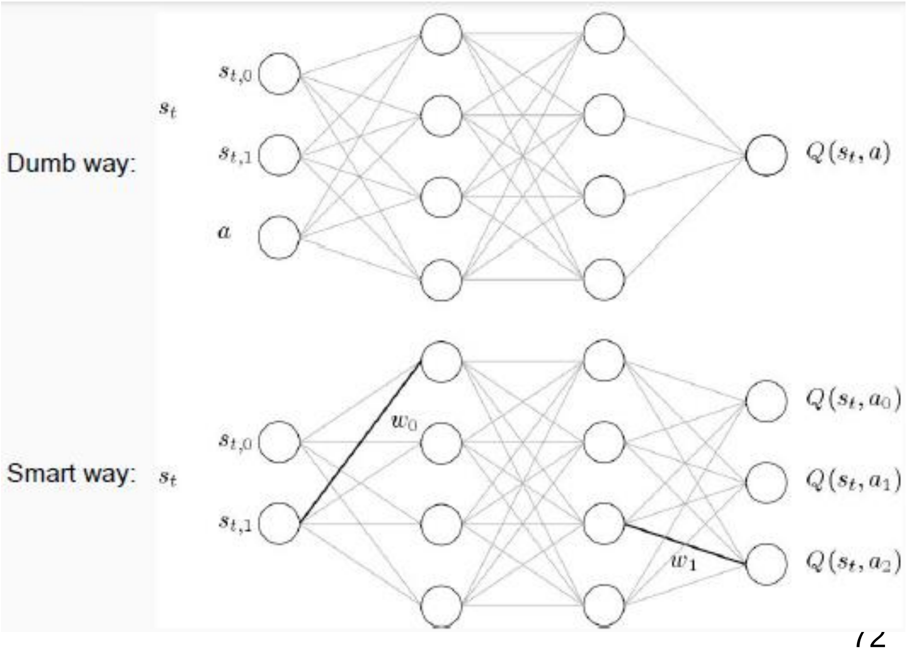
Deep Q Learning

https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

# Function Approximations of Value Functions

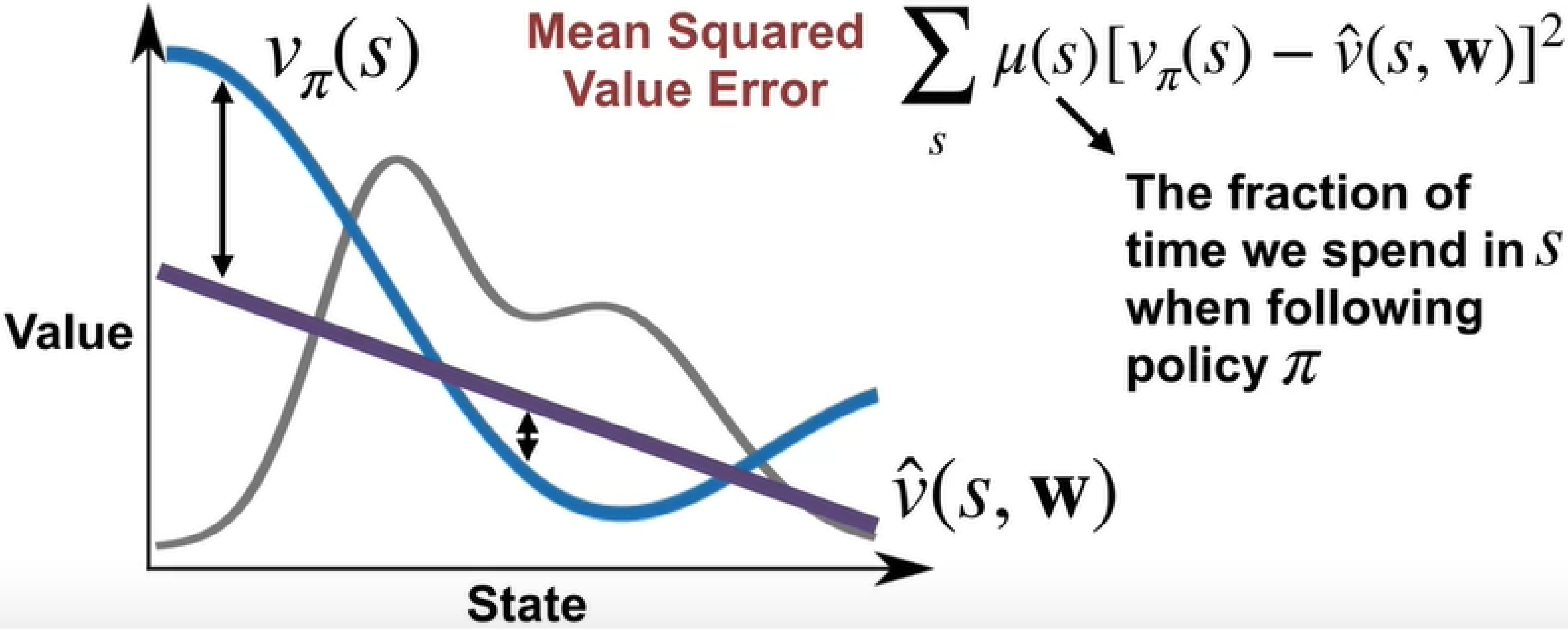
- Upper:
  - Left: state value function $\hat{v}(s, \mathbf{w})$ with params $\mathbf{w}$.
  - Middle: action value function $\hat{q}(s, a, \mathbf{w})$ with params $\mathbf{w}$.
  - Right: action value functions $\hat{q}(s, a_i, \mathbf{w})$ with params $\mathbf{w}$, since we need all Q-values for computing greedy policy $\mathrm{argmax_a}\, Q(s, a)$.
- Lower:
  - Use Neural Network as action value functions (corresponds to upper middle and right).


$\hat{v}(s,\mathbf{w})$
$\hat{q}(s,a,\mathbf{w})$
$\hat{q}(s,a_1,\mathbf{w})$ ··· $\hat{q}(s,a_m,\mathbf{w})$
w
w
w
s
s
a
s



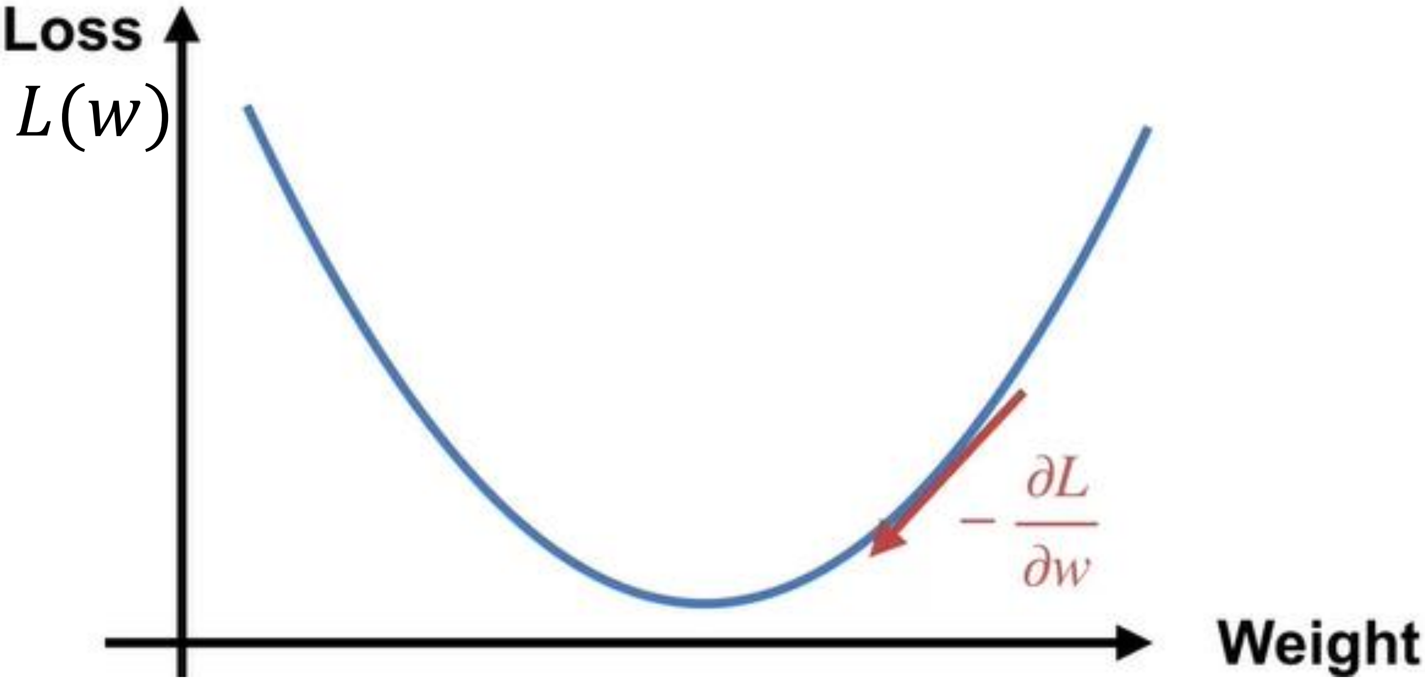
# Mean Squared Error

- Optimization objective is to minimize Value Error
  - $\overline{VE} = \mathbb{E}_\pi[v_\pi(s) - \hat{v}(s,\boldsymbol{w})]^2 = \sum_{s\in\mathcal{S}} \mu(s)[v_\pi(s) - \hat{v}(s,\boldsymbol{w})]^2$
  - $\mu(s)$ is the fraction of time spent in state $s$ when following policy $\pi$, called the on-policy distribution under policy $\pi$, $\sum_{s\in\mathcal{S}} \mu(s) = 1$. A larger $\mu(s)$ denotes state $s$ is visited more frequently, hence estimation error of $v_\pi(s)$ is given more weight



# Gradient Descent

- The $x$-axis corresponds to weight vector $\boldsymbol{w}$, and the $y$-axis to the objective value (i.e., loss function) $L(\boldsymbol{w})$ for weight $\boldsymbol{w}$.
- To minimize $L(\boldsymbol{w})$, we adjust weight vector $\boldsymbol{w}$ in the direction of the negative of the gradient $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla L(\boldsymbol{w})$
  - $\alpha$ is step-size parameter that is typically gradually reduced

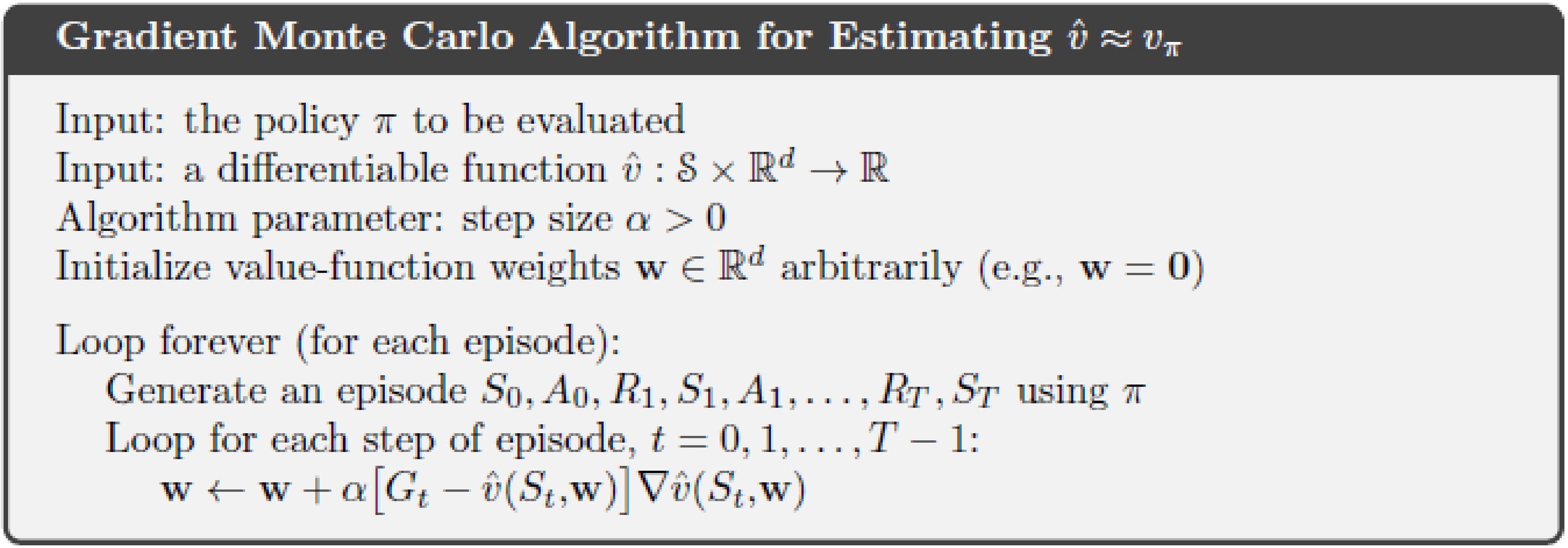


# Stochastic Gradient Descent (SGD)

- Gradient Descent for minimizing Mean Squared Value Error (impractical for a large number of states)
  - $\nabla\overline{VE} = \nabla \sum_{s\in\mathcal{S}} \mu_\pi(s)[v_\pi(s) - \hat{v}(s,\boldsymbol{w})]^2 = \sum_{s\in\mathcal{S}} \mu_\pi \nabla[v_\pi(s) - \hat{v}(s,\boldsymbol{w})]^2 = -2\sum_{s\in\mathcal{S}} \mu_\pi [v_\pi(s) - \hat{v}(s,\mathbf{w})]\,\nabla\hat{v}(s,\boldsymbol{w})$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2}\alpha\nabla\overline{VE} = \boldsymbol{w} + \alpha\sum_{s\in\mathcal{S}} \mu_\pi [v_\pi(s) - \hat{v}(s,\mathbf{w})]\,\nabla\hat{v}(s,\boldsymbol{w})$
- SGD: on each step, update $\boldsymbol{w}$ based on a single new state $S_t$ and its value $v_\pi(S_t)$:
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[v_\pi(S_t) - \hat{v}(S_t,\boldsymbol{w})]\nabla\hat{v}(S_t,\boldsymbol{w})$

# Gradient Monte Carlo

- Gradient Monte Carlo: Use MC target $G_t$ as unbiased estimate of $v_\pi(S_t)$.
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[G_t - \hat{v}(S_t, \boldsymbol{w})]\nabla\hat{v}(S_t, \boldsymbol{w})$
  - Recall $G_t \doteq \sum_{k=0}^{T-1} \gamma^k R_{t+k+1}$

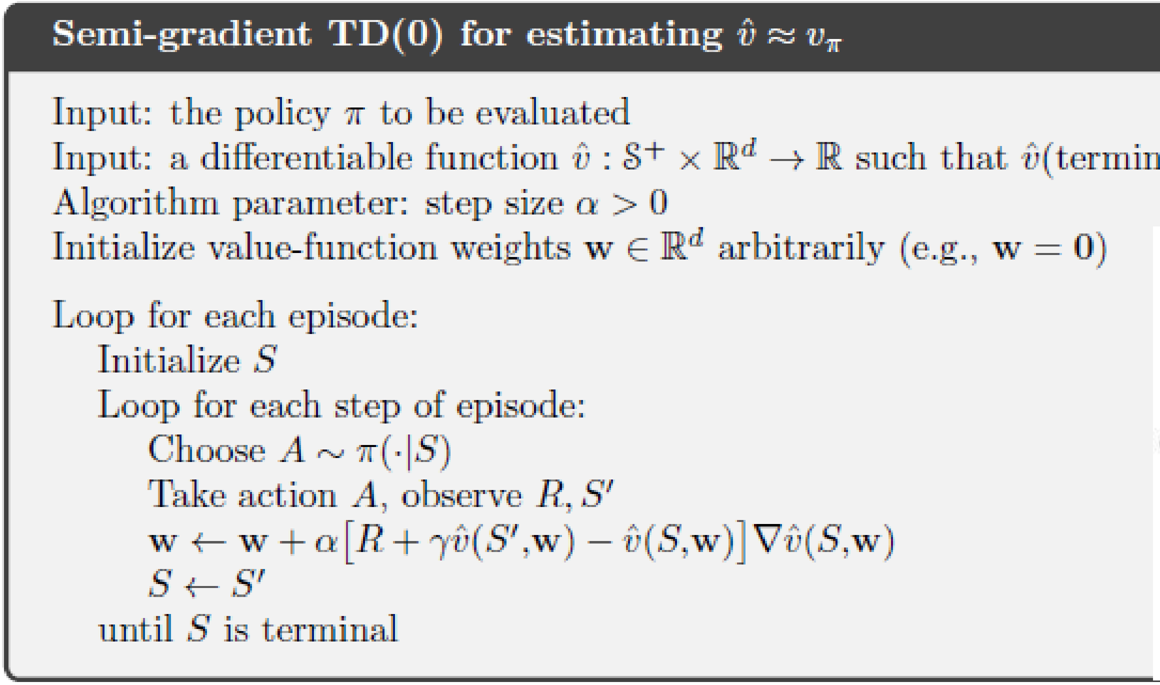


**Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$**

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using $\pi$
    Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$
```

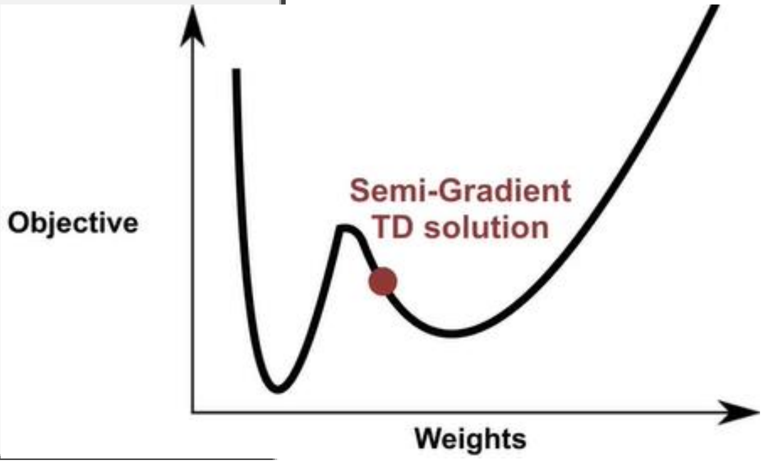
# Semi-Gradient TD(0) for Estimating $\hat{v} \approx v_\pi$

- Use TD target $R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w})$ as biased estimate of $v_\pi(S_t)$.
- $\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2}\alpha\nabla[R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})]^2 \neq \boldsymbol{w} + \alpha[R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})]\nabla\hat{v}(S_t, \boldsymbol{w})$
  - Since $\nabla\big(R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w})\big) \neq 0$
- Semi-Gradient TD(0): use the semi-gradient as approximation to the real gradient
  - $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$
- It may not converge to local minimum, but it converges faster than Gradient MC due to more frequent (per timestep instead of per episode) and less noisy updates.



**Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$**

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = 0$)

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action $A$, observe $R, S'$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until $S$ is terminal
```



# TD vs. Supervised Learning

- TD tries to learn parametrize value function $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$
  - Learn mapping $S_t \rightarrow R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$ from training dataset: $\{(S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w})), (S_2, R_3 + \gamma \hat{v}(S_3, \boldsymbol{w})), (S_3, R_4 + \gamma \hat{v}(S_4, \boldsymbol{w})),$ ... $\}$
  - Target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$ depends on $\boldsymbol{w}$. (non i.i.d)
- c.f. Supervised Learning, with fixed and given target (i.i.d)
  - Learn mapping $X_i \rightarrow \hat{y}(X_i, \boldsymbol{w})$ from training dataset: $\{(X_1, Y_1), (X_2, Y_2), (X_3, Y_3)$ ... $\}$

# TD w. Linear Function Approximation

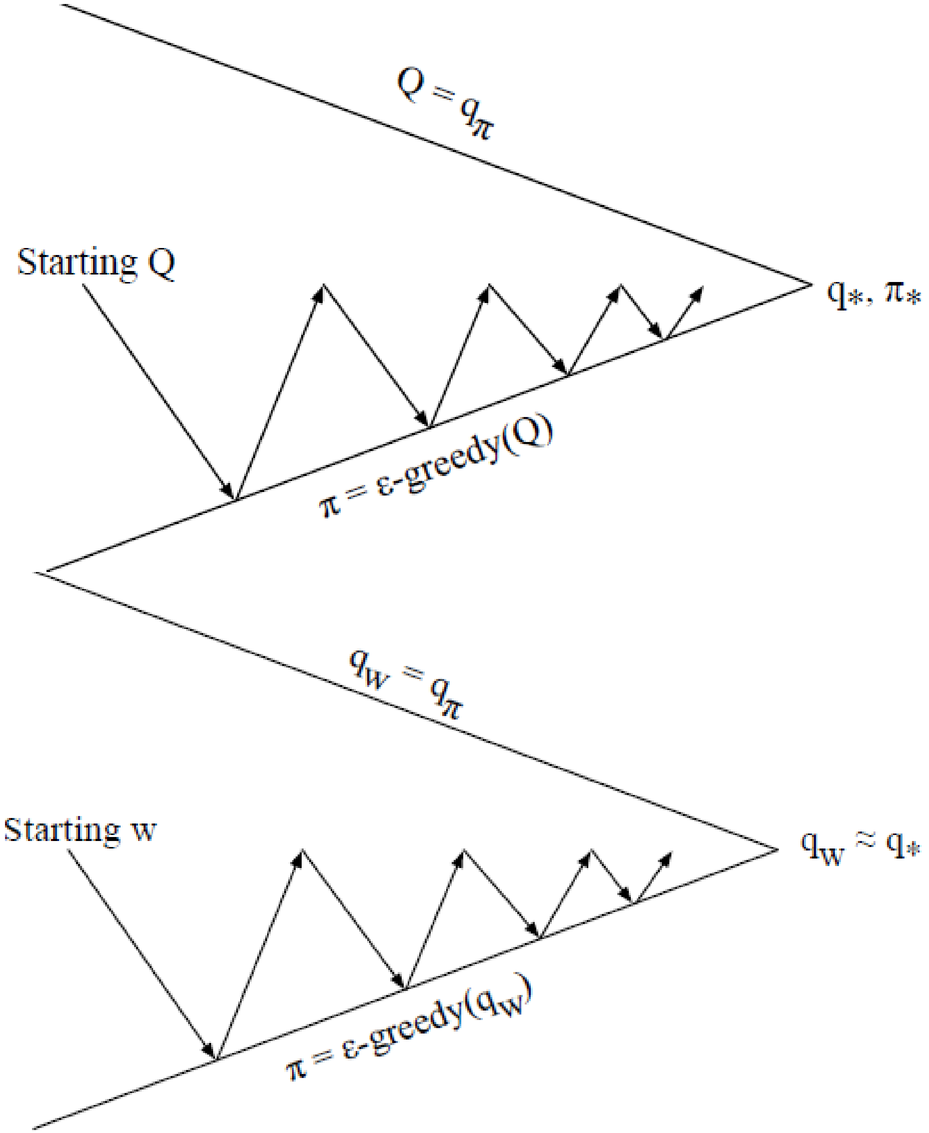
- For linear value function approximation $\hat{v}(s,\boldsymbol{w}) \doteq \boldsymbol{w}^T \boldsymbol{x}(s)$, we have $\nabla\hat{v}(s,\boldsymbol{w}) = \boldsymbol{x}(s)$
  - e.g. for feature vector of size 2:
  - $\hat{v}(s,\boldsymbol{w}) \doteq [w_1 \quad w_2] \begin{bmatrix} x_1(s) \\ x_2(s) \end{bmatrix} = w_1 x_1(s) + w_2 x_2(s)$
  - $\nabla\hat{v}(s,\boldsymbol{w}) = \begin{bmatrix} \frac{\partial \hat{v}(s,\mathbf{w})}{\partial w_1} \\ \frac{\partial \hat{v}(s,\mathbf{w})}{\partial w_2} \end{bmatrix} = \begin{bmatrix} x_1(s) \\ x_2(s) \end{bmatrix}$
- Semi-gradient TD:
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[R_{t+1} + \gamma\hat{v}(S_{t+1},\boldsymbol{w}) - \hat{v}(S_t,\boldsymbol{w})]\boldsymbol{x}(S_t)$
  - Weight update = step size × TD error × feature value
- A theorem relating Linear TD's fixed-point and minimum of Value Error:
  - $\overline{VE}(\boldsymbol{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{w}} \overline{VE}(\boldsymbol{w})$

# Tabular TD is a Special Case of Linear TD

- For Tabular TD, feature vector is one-hot encoding of states: $\boldsymbol{x}(s_i) = [0 \ \ldots 0 \; 1 \; 0 \ \ldots 0]^T$, with 1 for the $i$-th element and 0 for all others. This assigns a value to each individual state $\hat{v}(s_i, \boldsymbol{w}) = w_i$
  - e.g., for 2 states $s_1, s_2$, we have 2 features: $\boldsymbol{x}(s_1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \boldsymbol{x}(s_2) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
  - $\hat{v}(s_1, \boldsymbol{w}) \doteq \boldsymbol{w}^T \boldsymbol{x}(s_1) = [w_1 \quad w_2] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = w_1$
  - $\hat{v}(s_2, \boldsymbol{w}) \doteq \boldsymbol{w}^T \boldsymbol{x}(s_2) = [w_1 \quad w_2] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = w_2$
- Semi-gradient TD becomes:
  - $w_i \leftarrow w_i + \alpha[R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})] \cdot 1$
  - Same as Tabular TD: $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
- Similarly, tabular MC is a special case of Linear Gradient MC.

# CH10 On-policy Control with Approximation

- Top: Generalized Policy Iteration (GPI) for tabular setting, updating $Q(S_t, A_t)$ for each $(S_t, A_t)$ in each iteration.
- Bottom: GPI for function approximation setting, updating $\boldsymbol{w}$ in $\hat{q}(s, a, \boldsymbol{w})$ for any $(s, a)$ in each iteration


$Q = q_\pi$
Starting Q
$q_*, \pi_*$
$\pi = \varepsilon\text{-greedy}(Q)$
$q_w = q_\pi$
Starting w
$q_w \approx q_*$
$\pi = \varepsilon\text{-greedy}(q_w)$

# Action Value Function Update Equations

- Recall Sarsa:
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
- Semi-Gradient Sarsa w. function approximation:
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \boldsymbol{w}))\nabla\hat{q}(S_t, A_t, \boldsymbol{w})$
- Recall Expected Sarsa:
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t))$
- Semi-Gradient Expected Sarsa w. Function Approximation:
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) \hat{q}(S_{t+1}, a', \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w}))\nabla\hat{q}(S_t, A_t, \mathbf{w})$
- Recall QL:
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t))$
- Semi-Gradient QL w. Function Approximation:
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w}))\nabla\hat{q}(S_t, A_t, \boldsymbol{w})$
- (Optional) linear function approximation $\hat{q}(s, a, \boldsymbol{w}) \doteq \boldsymbol{w}^T \boldsymbol{x}(s, a)$