Lecture 2: Tabular RL Algorithms

Chunlin Chen & Zhi Wang

Department of Control and Systems Engineering
Nanjing University

Nov. 12th, 2019

Table of Contents

Dynamic Programming (DP)

- A collection of algorithms that can be used to compute optimal policies given a perfect model of the environment (MDP)
  - Of limited utility in RL both because of their assumption of a perfect model and because of their great computational expense
  - Important theoretically, provide an essential foundation for the understanding of RL methods
  - RL methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment

Policy evaluation (Prediction)

- Compute the state-value function v_π for an arbitrary policy π

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

- If the environment's dynamics are completely known
  - In principal, the solution is a straightforward computation

Iterative policy evaluation

- Consider a sequence of approximate value functions $v_0, v_1, v_2, ...$
  - The initial approximation, v_0, is chosen arbitrarily
- Use the **Bellman equation** for v_π as an update rule

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned}$$

- $v_k = v_\pi$ is a fixed point for this update rule
  - The sequence $\{v_k\}$ converges to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π

Iterative policy evaluation

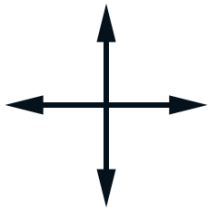
Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

- The updates as being done in a **sweep** through the state space

Example: Gridworld



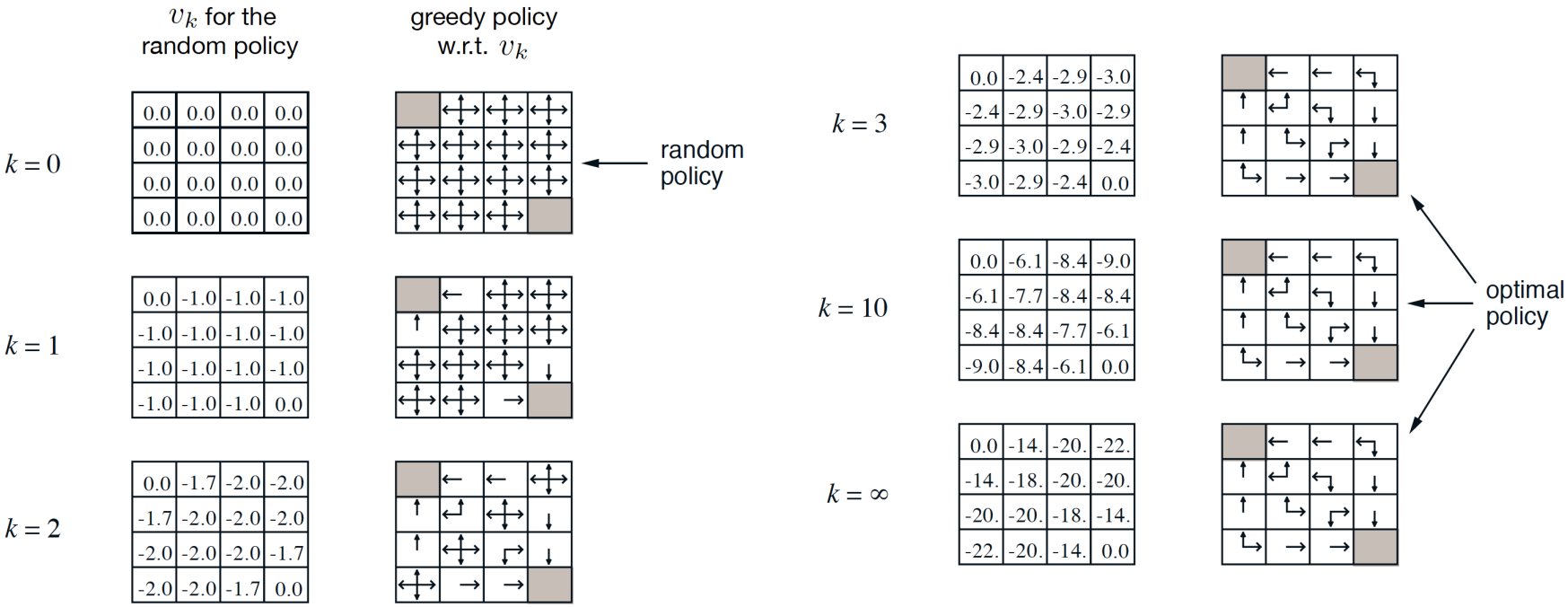
actions

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions

- Four actions deterministically case the corresponding state transitions
  - e.g., $p(6, -1|5, right) = 1, p(7, -1|7, right) = 1$
  - $p(10, r|5, right) = 0, \forall r \in \mathcal{R}$

The agent follows the equiprobable random policy



- The final estimate is in fact v_π
  - The negation of the expected number of steps from that state until termination
  - The last policy is guaranteed only to be an improvement over the random policy, but in this case it, and all policies after the third iteration, are optimal

Policy improvement

- Our reason for computing the value function for a policy is to help find better policies
  - We have determined the value function v_π for policy π
  - we would like to know whether or not we should change the policy to deterministically choose an action $a \neq \pi(s)$
  - We know how good it is to follow the current policy from s, e.g., v_π, but would it be better or worse to change to the new policy, π'?

- Consider selecting a in s and thereafter following the existing policy π

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r | s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

  - If $Q_\pi(s,a) \geq v_\pi(s)$?

Policy improvement theorem

- Let π and π' be any pair of deterministic policies such that,

$$Q(s, \pi'(s)) \geq v_\pi(s), \quad \forall s \in \mathcal{S}.$$

Then the policy π' must be as good as, or better than, π.

Policy improvement theorem

$$
\begin{aligned}
v_\pi(s) &\leq Q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma Q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s] \\
&\leq ... \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... | S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}
$$

Policy improvement

- Consider the new **greedy** policy, π', selecting at each state the action that appears best according to $Q_\pi(s, a)$

$$\begin{aligned}\pi'(s) &= \arg\max_a Q_\pi(s, a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \arg\max_a \sum_{s', a} p(s', r | s, a)[r + \gamma v_\pi(s')]\end{aligned}$$

- The process of making a new policy that improves on an original policy, by making greedy w.r.t. the value function of the original policy, is called **policy improvement**
  - The greedy policy meets the conditions of the policy improvement theorem

Policy improvement

- Suppose the new policy π' is as good as, but not better than, the old policy π, then $v_{\pi'} = v_\pi$

$$
\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r | s, a)[r + \gamma v_{\pi'}(s')]
\end{aligned}
$$

  - The sane as the Bellman optimality equation
  - Both π and π' must be optimal policies
- Policy improvement must give us a strictly better policy except when the original policy is already optimal

Policy iteration

- Using policy improvement theorem, we can obtain a sequence of monotonically improving policies and value functions

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

- This process is guaranteed to converge to an optimal policy and optimal value function in a finite number of iterations
  - Each policy is guaranteed to be a strictly improvement over the previous one unless it is already optimal
  - A finite MDP has only a finite number of policies

Policy iteration

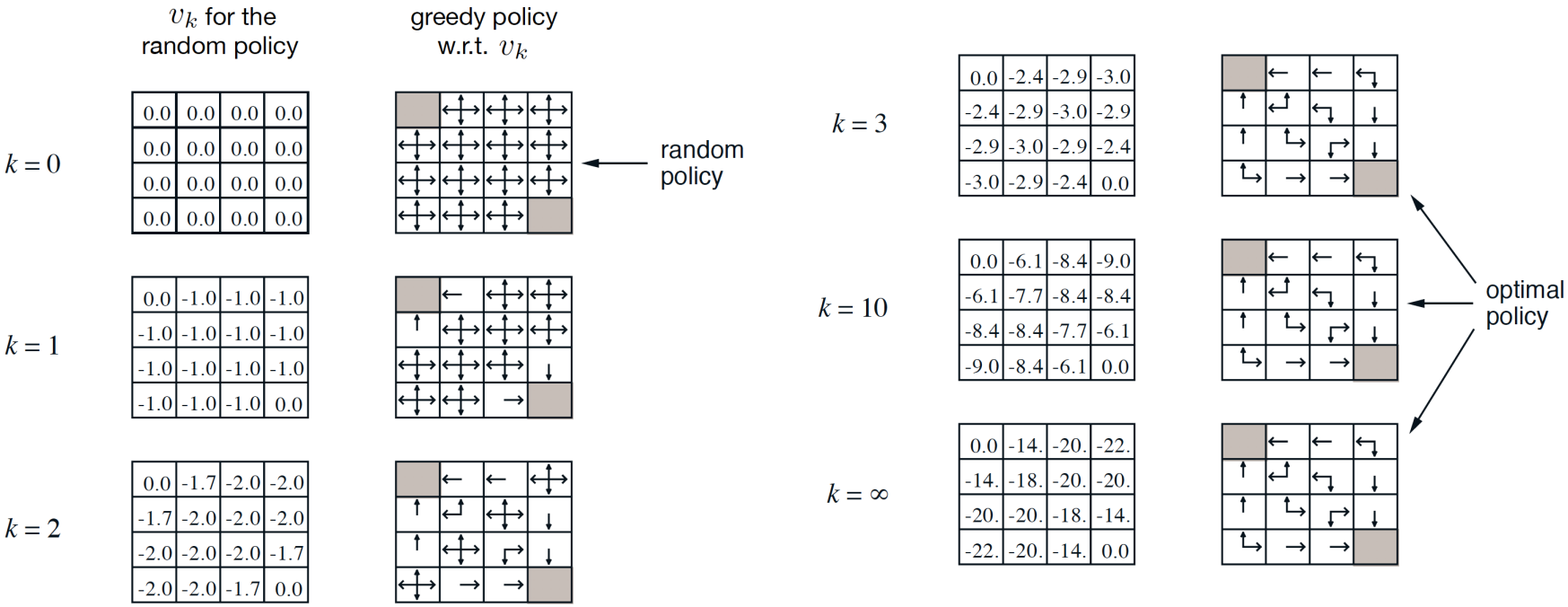
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-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-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy iteration often converges in very few iterations



- The final estimate is in fact v_π
  - The negation of the expected number of steps from that state until termination
  - The last policy is guaranteed only to be an improvement over the random policy, but in this case it, and all policies after the third iteration, are optimal

Value iteration

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

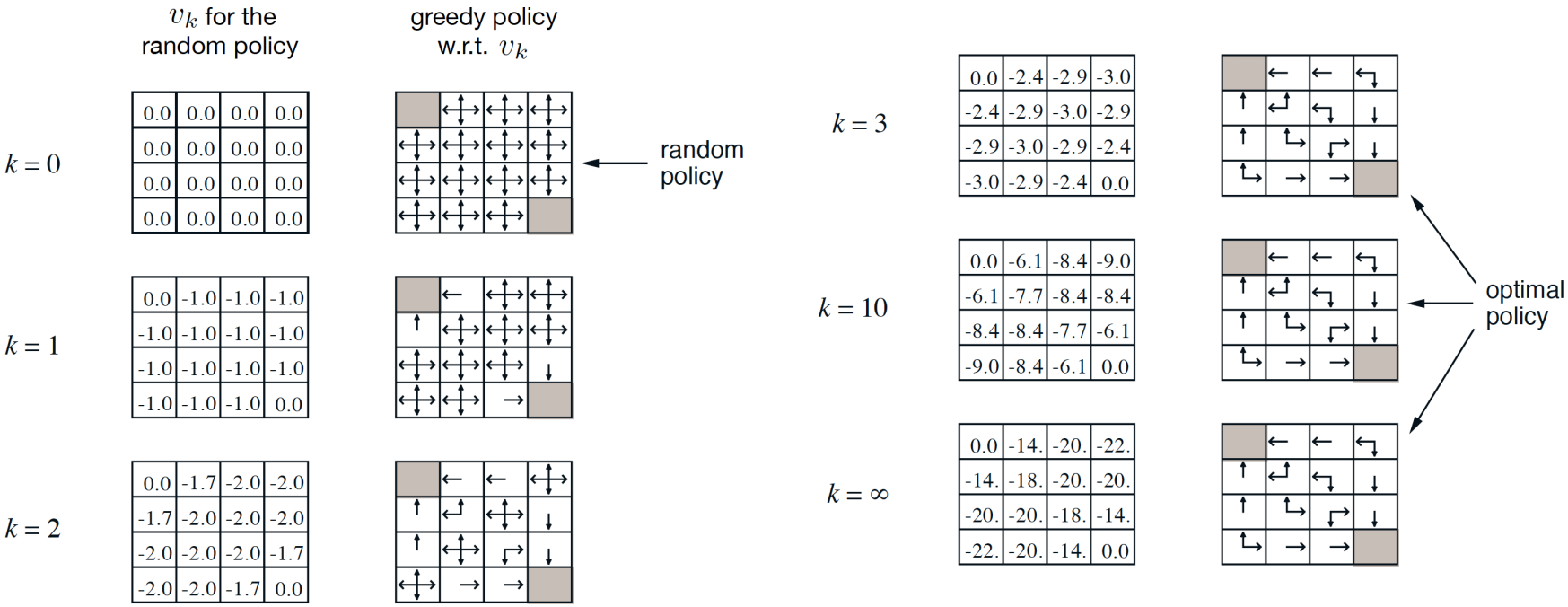
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

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

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

- Each policy iteration involves policy evaluation, which may be a protracted iterative computation **requiring multiple sweeps through the state set**

Truncate policy evaluation?



- Policy evaluation iterations beyond the first three have no effect on the corresponding greedy policy

Value iteration = Truncate policy evaluation for one sweep

- In policy iteration, stop policy evaluation after just one sweep

$$v_{k+1}(s) = \sum_{s',r} p(s', r|s, \pi_k(s))[r + \gamma v_k(s')]$$

$$\pi_{k+1}(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_{k+1}(s')]$$

- Combine into one operation, called **value iteration** algorithm

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

  - For arbitrary v_0, the sequence $\{v_k\}$ converges to v_* under the same conditions that guarantee the existence of v_*

Value iteration

- Bellman optimality equation

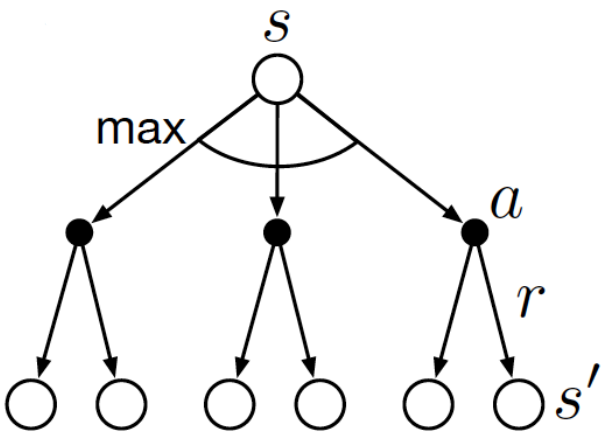
$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

- Value iteration

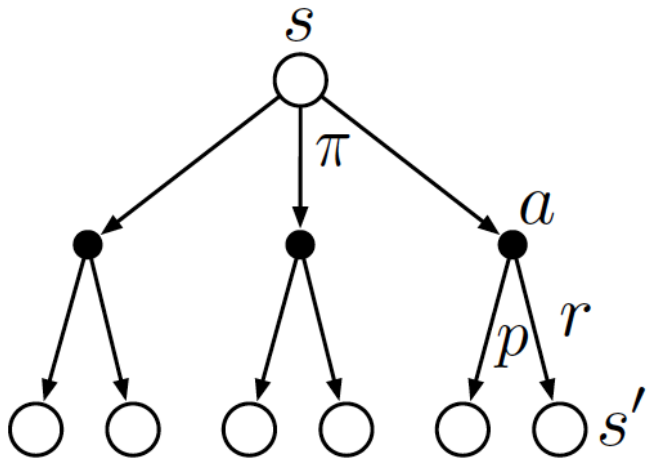
$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

  - Turn Bellman optimality equation into an update rule
  - Directly approximate the optimal state-value function, v_*

Value iteration vs. policy evaluation



Backup diagram for value iteration



Backup diagram for policy evaluation

Value iteration algorithm

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

- One sweep = one sweep of policy evaluation + one sweep of policy improvement

Properties of dynamic programming

- **Bootstrapping**: Update estimates on the basis of other estimates
  - Estimate the values of states based on estimates of the values of successor states
- **Model-based**: Require the accurate model of the environment
  - The complete probability distributions of all possible transitions, $p(s', r|s, a)$

Algorithms	Bootstrapping?	Model-based?
Dynamic programming	Yes	Yes
Monte Carlo methods	No	No
Temporal-difference learning	Yes	No

Research outputs using DP

IEEE/ASME TRANSACTIONS ON MECHATRONICS, VOL. 24, NO. 2, APRIL 2019 621

Incremental Reinforcement Learning With Prioritized Sweeping for Dynamic Environments

Zhi Wang, Chunlin Chen, *Member, IEEE*, Han-Xiong Li, *Fellow, IEEE*, Daoyi Dong, *Senior Member, IEEE*, and Tzyh-Jong Tarn, *Life Fellow, IEEE*

Algorithm 3: Prioritized Sweeping of Drift Environment.

Input: $Q^*(s,a), \forall (s,a) \in (S,A)$ in E;
the small threshold θ
Output: $Q_{dn}(s_{dn}, a_{dn}), \forall (s_{dn}, a_{dn}) \in (S_{dn}, A_{dn})$

$E_d \leftarrow$ the drift environment using Algorithm 2
$E^m_{dn} \leftarrow m$-degree neighbor environment
Initialize E^m_{dn} : $Q_{dn}(s_{dn}, a_{dn}) \leftarrow Q^*(s_{dn}, a_{dn})$, $\forall (s_{dn}, a_{dn}) \in (S_{dn}, A_{dn}) \bigcap (S, A)$
Initialize: $\Delta_{max} \leftarrow \infty$
while $\Delta_{max} \geq \theta$ **do**
  $\Delta_{max} \leftarrow 0$
  for $(s_{dn}, a_{dn}) \in (S_{dn}, A_{dn})$ **do**
    $Q_{temp} \leftarrow \sum_{s'_{dn}} p(s'_{dn}|s_{dn}, a_{dn})[$
    $r_{dn}(s_{dn}, a_{dn}, s'_{dn}) + \gamma \max_{a'_{dn}} Q_{dn}(s'_{dn}, a'_{dn})]$
    $\Delta \leftarrow |Q_{temp} - Q_{dn}(s_{dn}, a_{dn})|$
    $Q_{dn}(s_{dn}, a_{dn}) \leftarrow Q_{temp}$
    if $\Delta > \Delta_{max}$ **then**
      $\Delta_{max} \leftarrow \Delta$
    end
  end
end

After updating the drift environment, we then update the state-action space of its m-degree neighbor environment with the second priority. It can be viewed as a process of spreading the changes caused by the drift environment to its neighbors and even to the whole state-action space gradually. In this process, the new information is fused into the existing knowledge system starting from the drift environment. The process of updating the state-action space of the drift environment and its neighbor environment with priority by dynamic programming is called as *prioritized sweeping of drift environment*, which is shown as in Algorithm 3.

Table of Contents

Model-based vs. Model-free

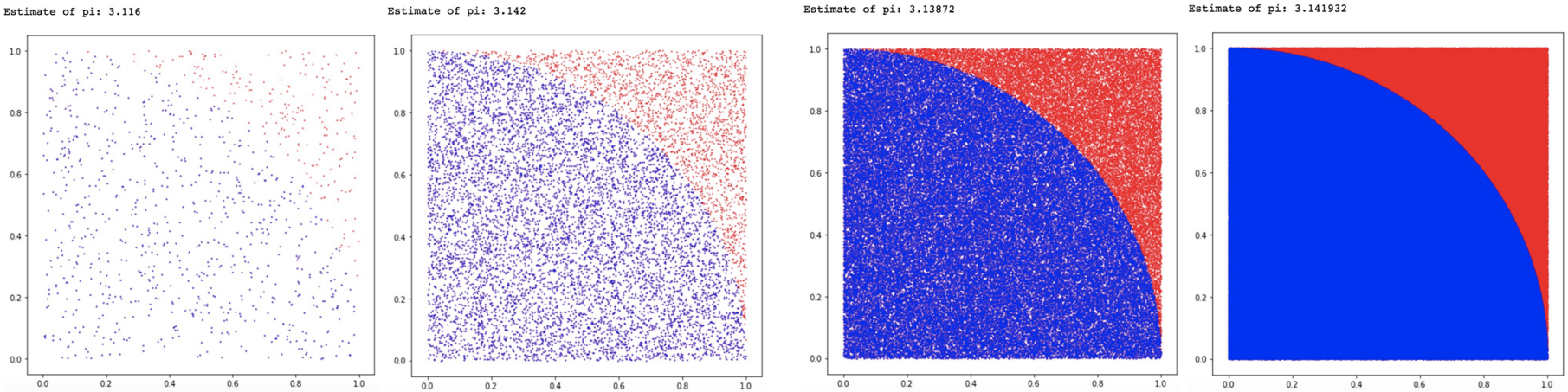
- Model-based algorithms, e.g., dynamic programming
  - Require the prior knowledge of the environment's dynamics, the complete probability distributions of all possible transitions
  - In many cases, it is easy to generate experience sampled according to the desired probability distributions, but **infeasible to obtain the distributions in explicit form**
- Model-free algorithms, e.g., Monte Carlo methods, temporal-difference learning
  - Require only **experience** – sample sequences of states, actions, and rewards from actual or simulated interaction with an environment
  - **Learning** from actual experience is striking because it requires no prior knowledge of the environment's dynamics, yet can still attain optimal behavior

The term "Monte Carlo" (MC)

- A broad class of computational algorithms that rely on repeated random sampling to obtain numerical results
  - Use randomness to solve problems that might be deterministic in principle
- The **Law of Large Numbers**: the basis for Monte Carlo simulations
  - As the number of identically distributed, randomly generated variables increases, their sample mean approaches their theoretical mean

Monte Carlo estimate of PI

- We can estimate pi to as many digits as we like by simply playing a game of darts
  - Generate random points within a box, and counting the number of points which fall within an embedded circle



Monte Carlo methods for RL

- Based on averaging sample returns
  - We define Monte Carlo methods only for episodic tasks, to ensure that well-defined returns are available
  - Incremental in an episode-by-episode sense, but not in a step-by-step sense
  - Average complete returns, as opposed to methods that learn from partial returns, e.g., temporal-difference learning

Monte Carlo prediction

- Considering Monte Carlo methods for learning the state-value function for a given policy
  - $v_\pi(s)$: the expected return–expected cumulative future discounted reward–starting from s
  - Estimate $v_\pi(s)$ from **experience**: simply average the returns observed after visits to s
  - As more returns are observed, the average should converge to the expected value

Monte Carlo prediction

- Some notations
  - Each occurrence of state s in an episode is called a **visit** to s
  - s may be visited multiple times in the same episode, we call the first time it is visited in an episode the **first visit** to s
- First-visit Monte Carlo method
  - Estimate $v_\pi(s)$ as the average of the returns following first visits to s
  - Have been most widely studied
- Every-visit Monte Carlo method
  - Average the returns following visits to s
  - Extend more naturally to function approximation and eligibility traces

First-visit MC prediction

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 $Returns(S_t)$

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

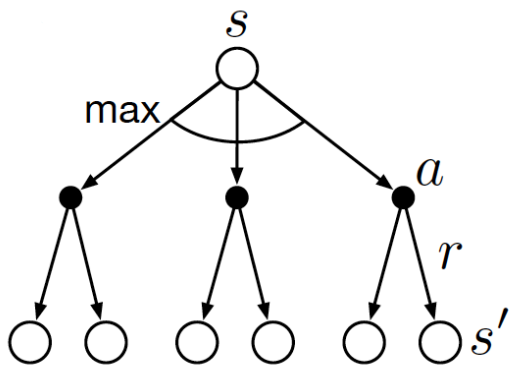
First-visit MC prediction

- Both first-visit MC and every-visit MC converge to v_π as the number of (first) visits to s goes to infinity
- For first-visit MC
  - Each return is an independent, identically distributed estimate of v_π with finite variance
  - Each average is an unbiased estimate, and the standard deviation of its error falls as $1/\sqrt{n}$

Backup diagram for MC prediction

- The root is a state node, and below it is the entire trajectory of transitions along a particular single episode, ending at the terminal state

- MC shows only those sampled on the one episode
  - Opposed to DP that shows all possible transitions
- MC goes all the way to the end of the episode
  - Opposed to DP that includes only one-step transitions



Properties of MC

- MC methods do not bootstrap
  - Estimates for each state are independent
  - The estimate for one state does not build upon the estimate of any other state
- The computational expense of estimating the value of a single state is independent of the number of states
  - This can make MC particularly attractive when one requires the value of only one or a subset of states
  - One can generate many sample episodes starting from the states of interest, averaging returns from only these states, ignoring all others

MC estimation of action values

- With a model
  - State values alone are sufficient to determine a policy
- Without a model
  - One must explicitly estimate the value of each action in order for the values to useful in suggesting a policy

- A state-action pair (s, a) is said to be visited in an episode if ever the state s is visited and action a is taken in it
  - To help in choosing among the actions available in each state, we need to estimate the value of all the actions from each state, not just the one we currently favor
  - This is the general problem of **maintaining exploration**

Maintaining exploration

- The assumption of **exploring starts**
  - Specify that the episodes start in a state-action pair, and that every pair has a nonzero probability of being selected as the start
  - Guarantee that tall state-action pairs will be visited an infinite number of times in the limit of an infinite number of episodes
  - Sometimes useful, but cannot be relied upon in general, particularly when learning directly from actual interaction with an environment

- The most common alternative approach to assuring that all state-action pairs are encountered is to consider only policies that are **stochastic** with a nonzero probability of selecting all actions in each state

Monte Carlo control

- Construct each π_{k+1} as the greedy policy w.r.t. Q_{π_k}

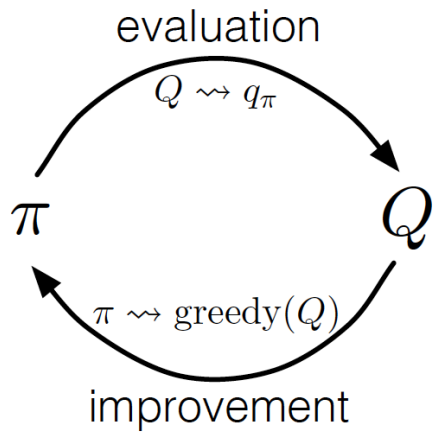
$$\pi_{k+1}(s) = \arg\max_a Q_{\pi_k}(s, a)$$

- Policy improvement theorem

$$\begin{aligned} Q_{\pi_k}(s, \pi_{k+1}(s)) &= Q_{\pi_k}(s, \arg\max_a Q_{\pi_k}(s, a)) = \max_a Q_{\pi_k}(s, a) \\ &\geq Q_{\pi_k}(s, \pi_k(s)) = v_{\pi_k}(s) \end{aligned}$$

- Use **generalized policy iteration** to approach optimal policies

$$\pi_0 \xrightarrow{E} Q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} Q_*$$



MC with policy iteration

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 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 $Returns(S_t, A_t)$

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

$\pi(S_t) \leftarrow \text{argmax}_a \, Q(S_t, a)$

- Two unlikely assumptions
  - The episodes have exploring starts
  - Policy evaluation could be done with an infinite number of episodes

MC control without exploring starts

- Without the assumption of exploring starts
  - If simply making the policy greedy w.r.t. the current value function, it can meet the policy improvement theorem, but would prevent further exploration of non-greedy actions
- Use a **soft** policy
  - $\pi(a|s) > 0, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
  - Gradually shifted closer and closer to a deterministic optimal policy
  - ε-soft policies: $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$
- ε-greedy policy
  - With probability ε, select an action at random
  - Otherwise, choose the greedy action
  - An example of the ε-soft policy, closest to greedy among ε-soft policies

MC control with soft polices

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

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

Policy improvement theorem for soft polices

- For any ε-soft policy, π, any ϵ-greedy policy, π', w.r.t. Q_π is guaranteed to be better than or equal to π

$$
\begin{aligned}
Q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) Q_\pi(s,a) \\
&= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a Q_\pi(s,a) + (1-\varepsilon) \max_a Q_\pi(s,a) \\
&\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a Q_\pi(s,a) + (1-\varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} Q_\pi(s,a) \\
&= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a Q_\pi(s,a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a Q_\pi(s,a) + \sum_a \pi(a|s) Q_\pi(s,a) \\
&= v_\pi(s)
\end{aligned}
$$

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

Monte Carlo tree search (MCTS)[11,12] uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function[12]. The strongest current Go programs are based on MCTS, enhanced by policies that are trained to predict human expert moves[13]. These policies are used

MC in AlphaGo

RESEARCH ARTICLE

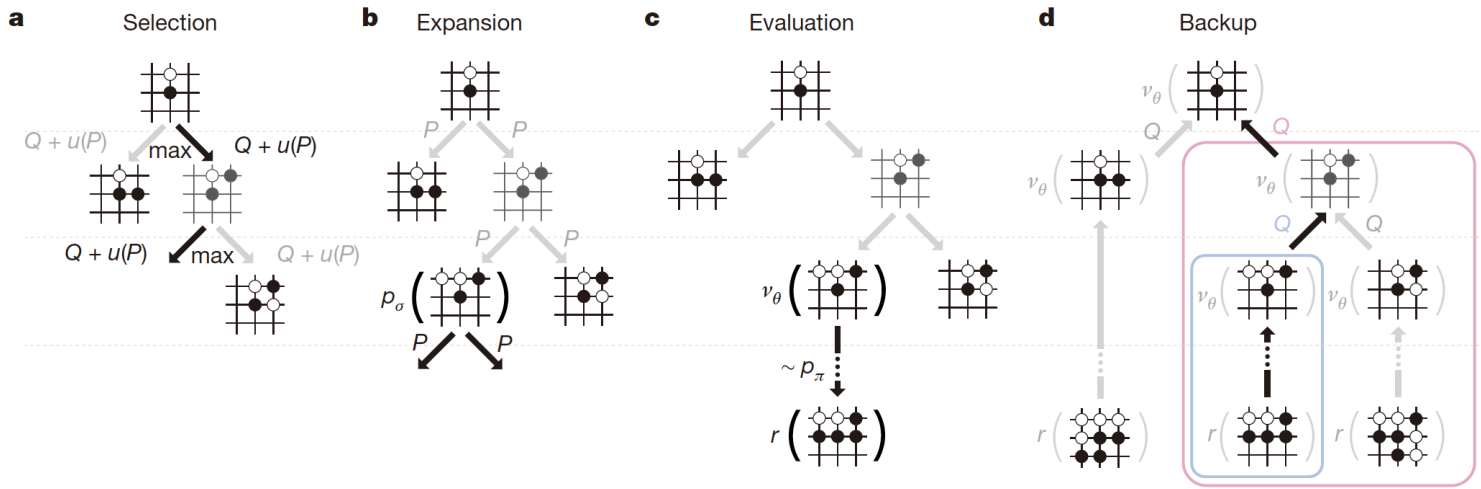


Figure 3 | Monte Carlo tree search in AlphaGo. **a**, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

- Instead of brute forcing from millions of possible ways to find the right path, Monte Carlo Tree Search algorithm chooses the best possible move from the current state of the game's tree with the help of RL.

Table of Contents

Temporal-Difference (TD) Learning

- If one had to identify one idea as central and novel to RL, it would undoubtedly be temporal-difference learning
  - A combination of Monte Carlo ideas and dynamic programming ideas
  - Like MC, TD can learn directly from raw experience without a model of the environment's dynamics (model-free)
  - Like DP, TD updates estimates based in part on other learned estimates, without waiting for a final outcome (bootstrap)

TD prediction

- MC and TD in common
  - Use experience to solve the prediction problem, update their estimate V of v_π for the non-terminal state S_t occurring in that experience

- MC: must wait until the return following the visit is known (end of an episode), then use that return as a target for $V(S_t)$

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

- TD: need to wait only until the next time step, use $R_{t+1} + \gamma V(S_{t+1})$ as the target for $V(S_t)$, bootstrapping

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

Basic TD algorithm

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 $V(terminal) = 0$

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

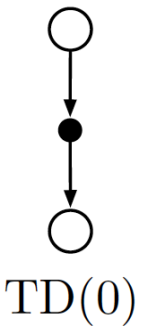
- **TD(0)**: A special case of the TD(λ) method using eligibility trace
- **One-step TD**: A special case of the n-step TD methods

Deep insight of the Bellman equation

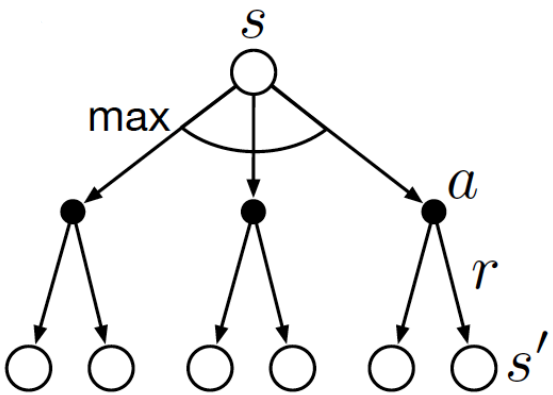
$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

- MC: The expected G_t is not known, a sample return is used in place of the real expected return
- DP: The true v_π is not known, and the current estimate $V(S_{t+1})$ is used instead
- TD: It samples the expected values R_{t+1}, and it uses the current estimate $V(S_{t+1})$ instead of the true v_π
  - Combine the sampling of MC with the bootstrapping of DP

Backup diagram for TD(0)



- The value estimate for the state node is updated on the basis of the one sample transition from it to the immediately following state
- **Sample updates**: based on a single sample successor, involve looking ahead to a sample successor state



- **Expected updates**: based on a complete distribution of all possible successors

TD error

- The difference between the estimated value of S_t and the better estimate $R_{t+1} + \gamma V(S_{t+1})$

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

- The MC error, with V not updated during the episode

$$\begin{aligned}
G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\
&= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\
&= \delta_t + \gamma\delta_{t+1} + \ldots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\
&= \delta_t + \gamma\delta_{t+1} + \ldots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(0 - 0) \\
&= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k
\end{aligned}$$

Advantages of TD prediction methods

- Compared to DP, do not require a model of the environment, of its reward and next-state probability distribution
  - In many cases, it is infeasible to obtain the distributions in explicit form
- Compared to MC, wait only one time step instead of waiting until the end of an episode
  - Applications have very long episodes, or continuing tasks have no episodes at all
- Are TD methods sound? Convergence guarantee?
  - It is convenient to learn one guess from the next, without waiting for an actual outcome
  - We can still guarantee convergence to the correct answer
  - For any fixed policy π, TD(0) has been proved to converge to v_π, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions

Convergence guarantee of TD prediction

- It is convenient to learn one guess from the next, without waiting for an actual outcome, and we can still guarantee convergence to the correct answer
  - For any fixed policy π, TD(0) has been proved to converge to v_π, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions

---

- Use the **Bellman equation** for v_π as an update rule

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned}$$

- $v_k = v_\pi$ is a fixed point for this update rule
  - The sequence $\{v_k\}$ converges to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π

TD control using action-value functions

Target policy $\pi(a\|s)$	Behavior policy $b(a\|s)$
To be evaluated or improved	To explore to generate data
Make decisions finally	Make decisions in training phase

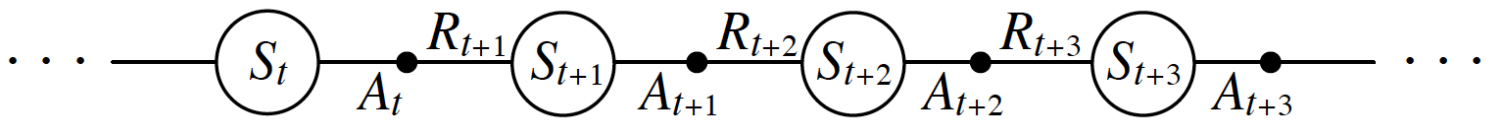
- **On-policy** methods: $\pi(a|s) = b(a|s)$
  - Evaluate or improve the policy that is used to make decisions during training
  - e.g., SARSA
- **Off-policy** methods: $\pi(a|s) \neq b(a|s)$
  - Evaluate or improve a policy different from that used to generate the data
  - Separate exploration from control
  - e.g., Q-learning

SARSA: On-policy TD control

- We follow the pattern of generalized policy iteration
  - Only this time using TD methods for the evaluation or prediction part
- Learn the values of state-action pairs instead of a state-value function
  - Consider transitions from state-action pair to state-action pair
- For an on-policy method, estimate $Q_\pi(s, a)$ for the current behavior policy π and for all state-action pairs

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

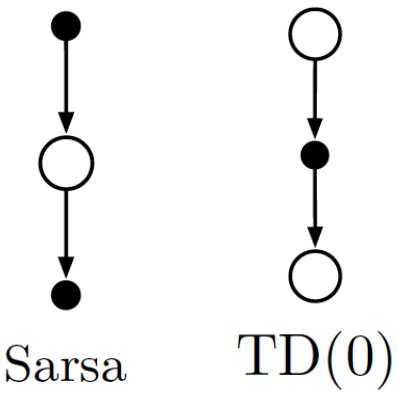
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$



SARSA: On-policy TD control

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

- This update is done after every transition from non-terminal state S_t
  - If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is defined as zero
  - Use every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$



- The theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm fro action values

SARSA algorithm

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., 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

- Follow the pattern of generalized policy iteration
  - Continually estimate Q_π for the behavior policy π, and at the same time change π toward greediness w.r.t. Q_π

Convergence properties of SARSA

- Let behavior policy π be greedy w.r.t. current action value function
  - Meet the policy improvement theorem
  - However, prevent further exploration of non-greedy actions
- Instead, **soft** policies are favored
  - ε-greedy or ε-soft policies

- Convergence properties of SARSA depend on the nature of the policy's dependence on the action-value function Q
  - SARSA converges with probability 1 to an optimal policy and action-value function as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (which can be arranged, for example, with ε-greedy policies by setting $\varepsilon = 1/t$)

Convergence properties of SARSA

- Use the **Bellman equation** for v_π as an update rule

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned}$$

- $v_k = v_\pi$ is a fixed point for this update rule
  - The sequence $\{v_k\}$ converges to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π

- Using policy improvement theorem, we can obtain a sequence of monotonically improving policies and value functions

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

- This process is guaranteed to converge to an optimal policy and optimal value function in a finite number of iterations
  - Each policy is guaranteed to be a strictly improvement over the previous one unless it is already optimal
  - A finite MDP has only a finite number of policies

Q-learning: Off-policy TD control

- One of the early breakthroughs in RL was the development of an off-policy TD control algorithm, known as **Q-learning**

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

- The learned action-value function, Q, directly approximates the optimal action-value function, Q_*, independent of the policy being followed

Q-learning: Off-policy TD control

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

- Use the Bellman optimality equation into an update rule

$$Q_*(S_t, A_t) = R_{t+1} + \gamma \max_a Q_*(S_{t+1}, a)$$

- Value iteration

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

  - Turn Bellman optimality equation into an update rule
  - Directly approximate the optimal state-value function, v_*

Q-learning algorithm

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

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

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

The reason for "off-policy"

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

- Q-learning approximates the optimal action-value function for an optimal policy, $Q \approx Q_* = Q_{\pi_*}$
  - The target policy is greedy w.r.t Q, $\pi(a|s) = \arg\max_a Q(s, a)$
  - The behavior policy can be others, e.g., $b(a|s) = \varepsilon$-greedy

---

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

- SARSA approximates the action-value function for the behavior policy, $Q \approx Q_\pi = Q_b$
  - The target and the behavior policy are the same, e.g., $\pi(a|s) = b(a|s) = \varepsilon$-greedy

Convergence properties of Q-learning

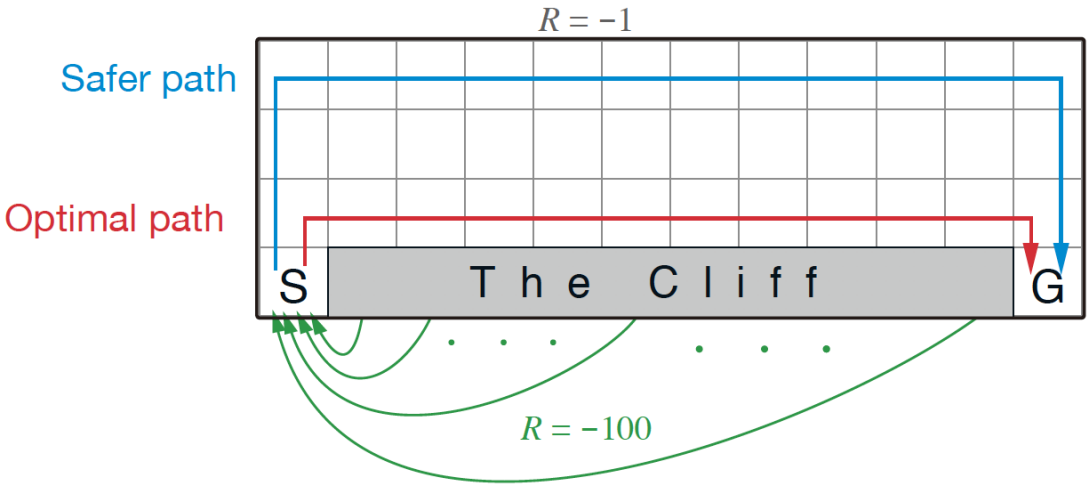
- All state-action pairs are visited an infinite number of times
  - This a minimal requirement in the sense that any method guaranteed to find optimal behavior in the general case must require it
  - Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-size parameters, Q has been shown to converge with probability 1 to Q_*

- Use the **Bellman equation** for v_π as an update rule

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned}$$

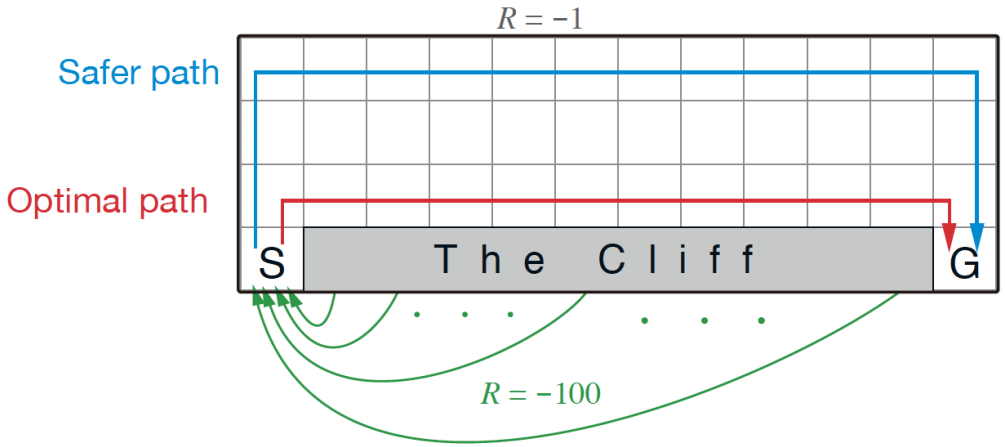
- $v_k = v_\pi$ is a fixed point for this update rule
  - The sequence $\{v_k\}$ converges to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π

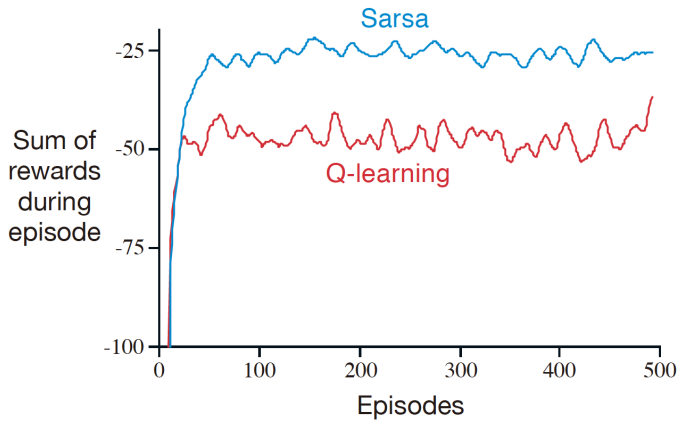
Example: Cliff walking



- The usual action causing movement up, down, right, and left
- $r = -1$ 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.
- With ε-greedy action selection, $\varepsilon = 0.1$

On-policy vs. Off-policy learning





- Q-learning learns values for the optimal policy, traveling right along the edge of the cliff
  - Occasionally fall off the cliff because of the ε-greedy action selection
- SARSA takes the action selection into account and learns the longer but safer path through the upper part of the grid
  - Q-learning's online performance is worse than that of SARSA
  - If ε is gradually reduced, both methods would asymptotically converge to the optimal policy

Table of Contents

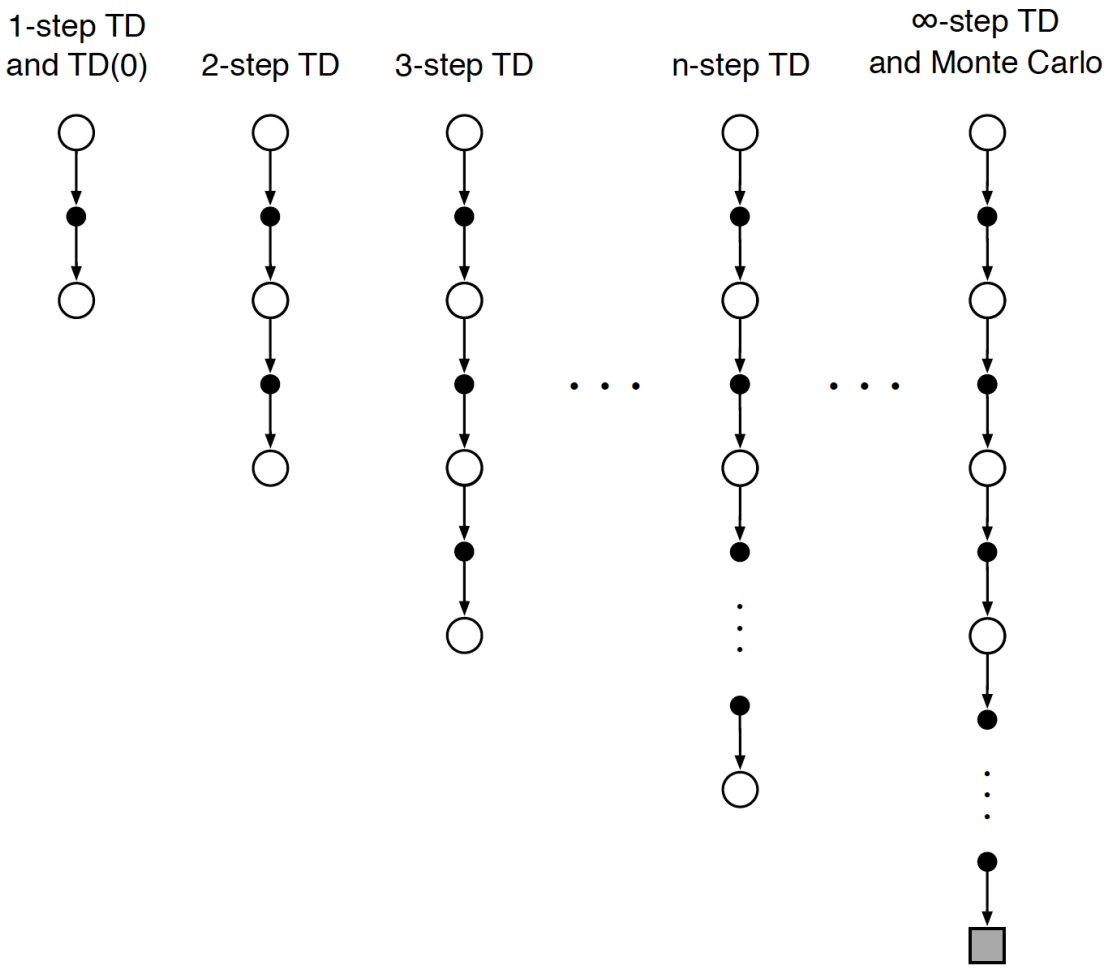
Combine MC and one-step TD

- Neither MC or one-step TD is always the best, we generalize both methods so that one can shift from one to the other smoothly as needed to meet the demands of a particular task
- One-step TD: In many applications, one wants to be able to update the action very fast to take into account anything that has changed
- However, bootstrapping works best if it is over a length of time in which a significant and recognizable state change has occurred

$n = 1$	n-step TD	$n = \infty$
TD(0)	$\longleftrightarrow$	MC

n-step TD prediction

- Perform an update based on an intermediate number of rewards, more than one, but less than all of them until termination



Recall MC and TD(0) updates

- In MC updates, the target is the **complete return**

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

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[G_t - V(S_t)] \\ &= V(S_t) + \alpha[R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t+1} R_T - V(S_t)] \end{aligned}$$

- In TD(0) updates, the target is the **one-step return**

$$G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1})$$

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[G_{t:t+1} - V(S_t)] \\ &= V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \end{aligned}$$

n-step TD update rule

- For n-step TD, set the target as the n-**step return**

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- All n-step returns can be considered approximations to the complete return, truncated after n steps and then corrected for the remaining missing terms by $V(S_{t+n})$

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[G_{t:t+n} - V(S_t)] \\ &= V(S_t) + \alpha[R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t)] \end{aligned}$$

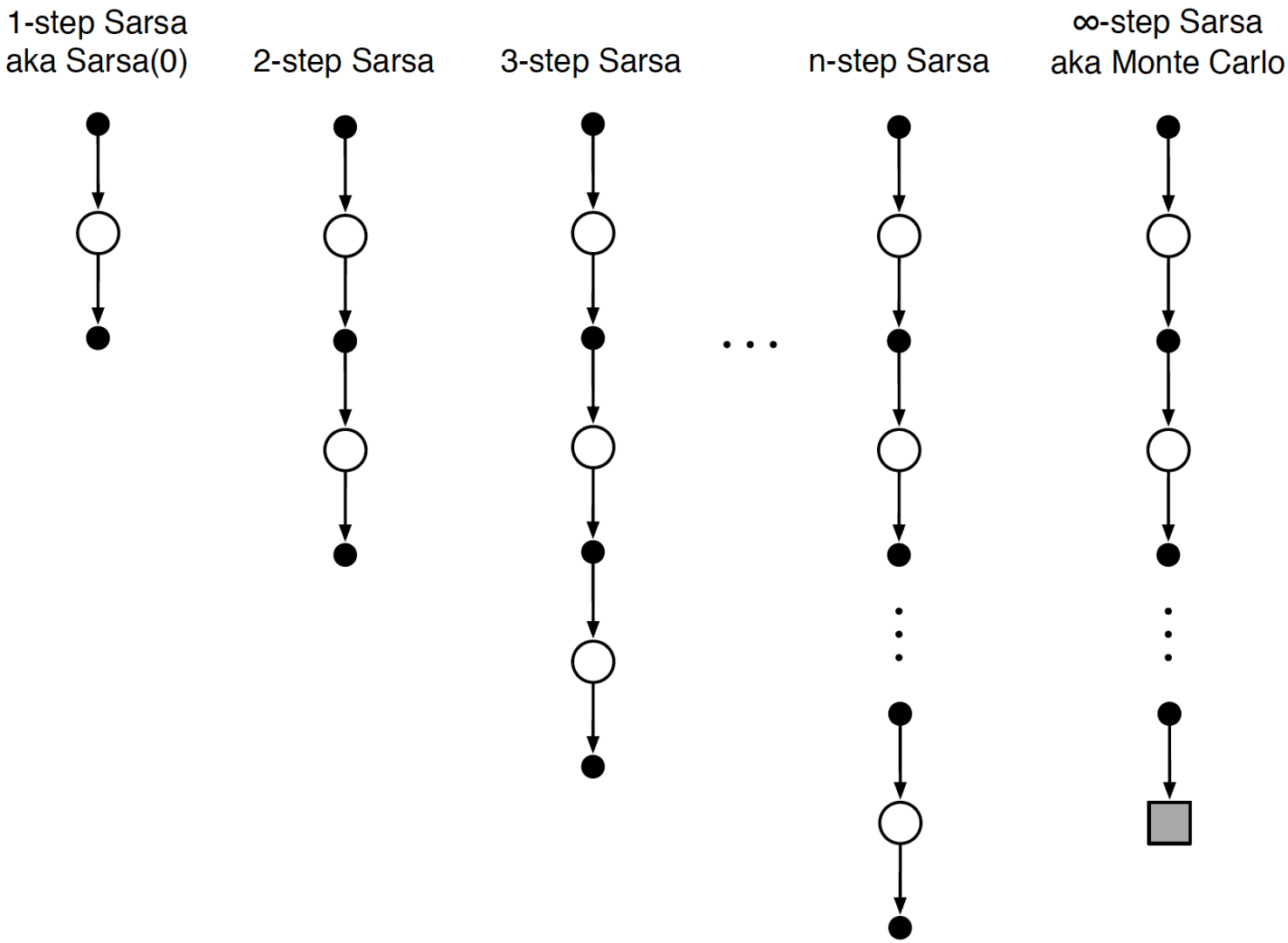
n-step TD algorithm

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 $\tau = T - 1$

n-step TD for on-policy control: n-step SARSA



From n-step TD prediction to n-step SARSA

- The main idea is to simply switch states for state-action pairs, and then use an ε-greedy policy
- re-define n-step returns (update targets) in terms of estimated action values

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha[G_{t:t+n} - Q(S_t, A_t)] \\ &= Q(S_t, A_t) + \alpha[R_{t+1} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n}) - Q(S_t, A_t)] \end{aligned}$$

n-step SARSA algorithm

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

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | | Take action A_t
  | | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | | If S_{t+1} is terminal, then:
  | | | $T \leftarrow t+1$
  | | else:
  | | | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
  | | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

Table of Contents

Eligibility traces: unify/generalize TD and MC

- Almost any TD method can be combined with eligibility traces to obtain a more general method that may learn more efficiently
  - e.g., the popular TD(λ) algorithm, λ refers the use of an eligibility trace
  - Produce a family of methods spanning a spectrum that has MC methods at one end ($\lambda = 1$) and one-step TD methods at the other ($\lambda = 0$)
- Eligibility traces offer an elegant algorithmic mechanism with significant computational advantages (compared to n-step TD)
  - Only a single trace vector is required rather than a store of the last n feature vectors
  - Learning also occurs continually and uniformly in time rather than being delayed and then catching up at the end of the episode
  - Learning can occur and effect behavior immediately after a state is encountered rather than being delayed n-steps

The λ-return

- How to interrelate TD and MC?
  - e.g., average one-step and infinite-step returns, $G = (G_t + G_{t:t+1})/2$
  - An update that averages simpler component updates is called a **compound update**
- The TD(λ) algorithm can be understood as one particular way of averaging n-step updates

$$
\begin{aligned}
G_t^\lambda &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n} \\
&= (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{T-t-1}G_t
\end{aligned}
$$

Backup diagram for TD(λ)

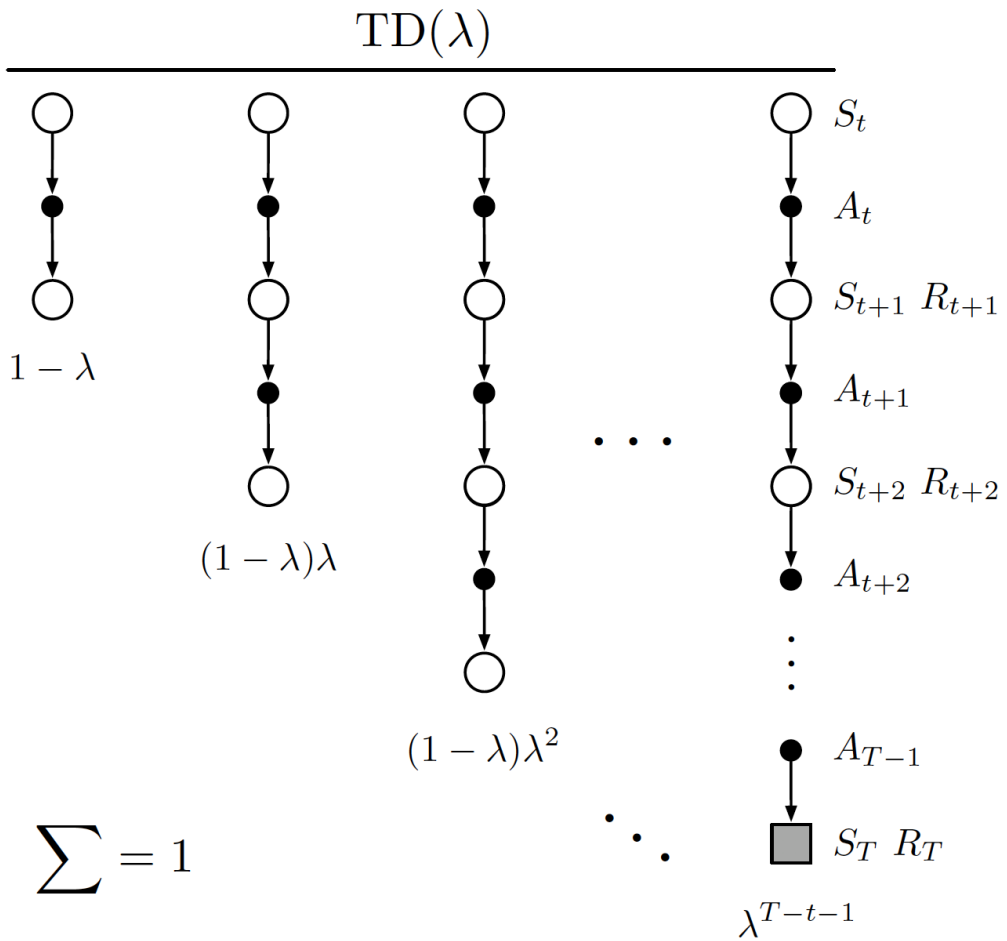


Figure 12.1: The backup digram for TD(λ). If $\lambda = 0$, then the overall update reduces to its first component, the one-step TD update, whereas if $\lambda = 1$, then the overall update reduces to its last component, the Monte Carlo update.

The weight distribution

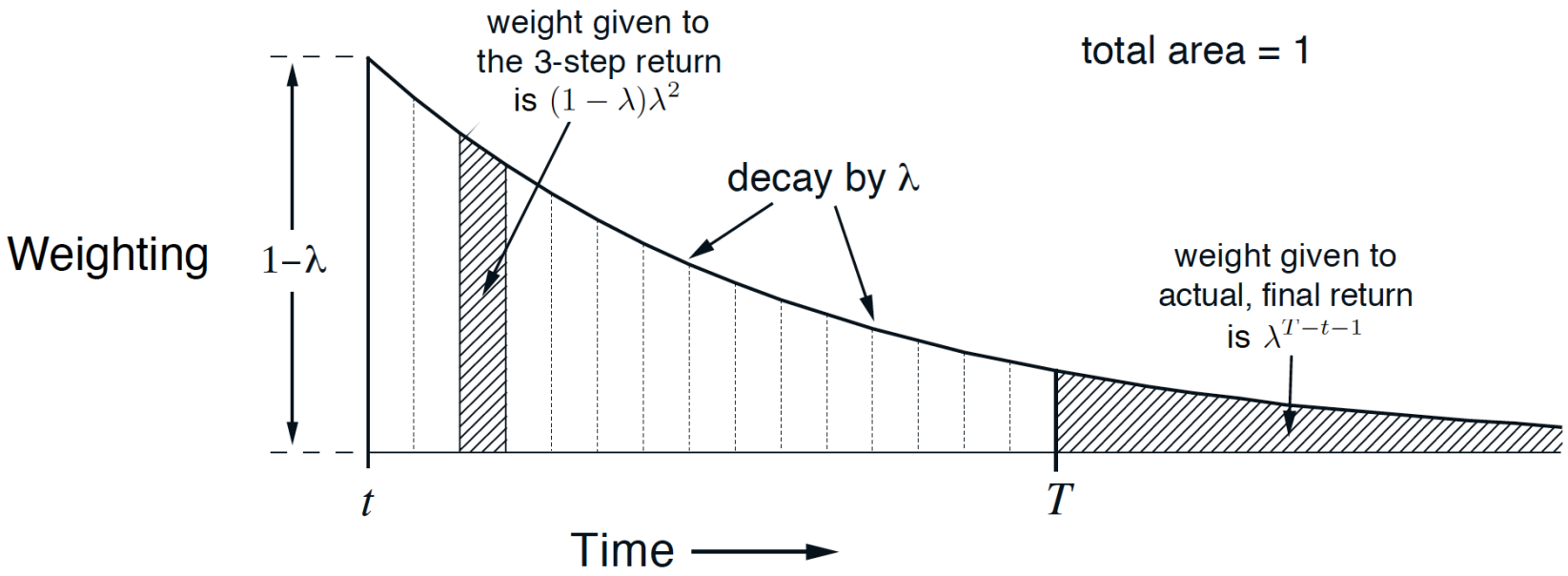


Figure 12.2: Weighting given in the λ-return to each of the n-step returns.

The forward view

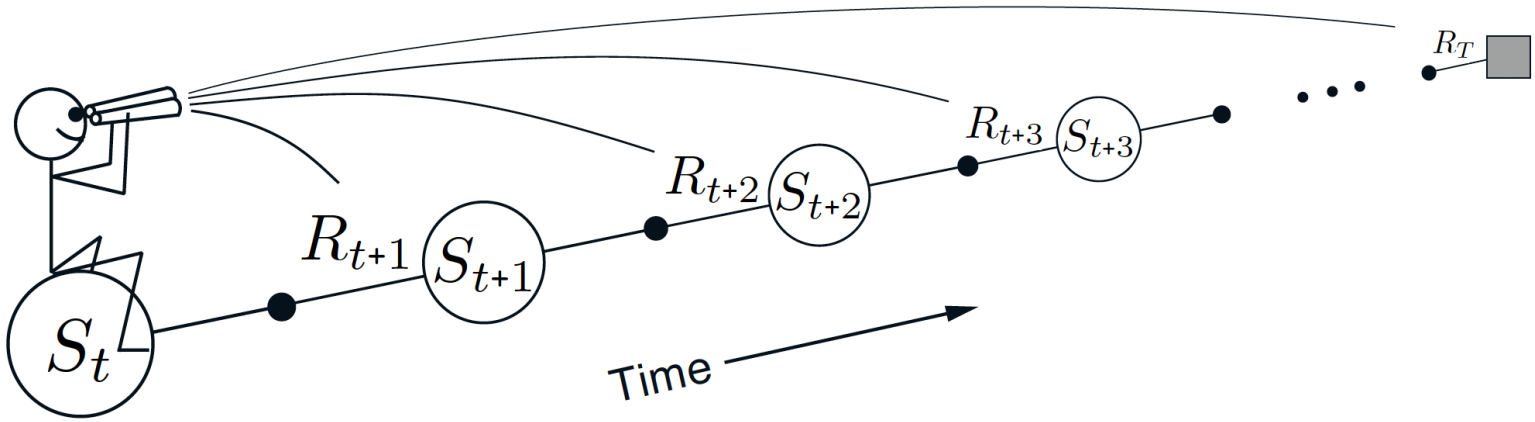


Figure 12.4: The forward view. We decide how to update each state by looking forward to future rewards and states.

TD(λ): distribute the computation in each time step

- The eligibility trace, $z(s), \forall s \in \mathcal{S}$

$$z_0(s) \leftarrow 0, \quad \forall s \in \mathcal{S}$$
$$z_t(s) \leftarrow \gamma\lambda z_{t-1}(s) + 1, \quad \forall s \in \mathcal{S}$$

- The update rule for state-value function

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

$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t \cdot z(S_t)$$

The forward view

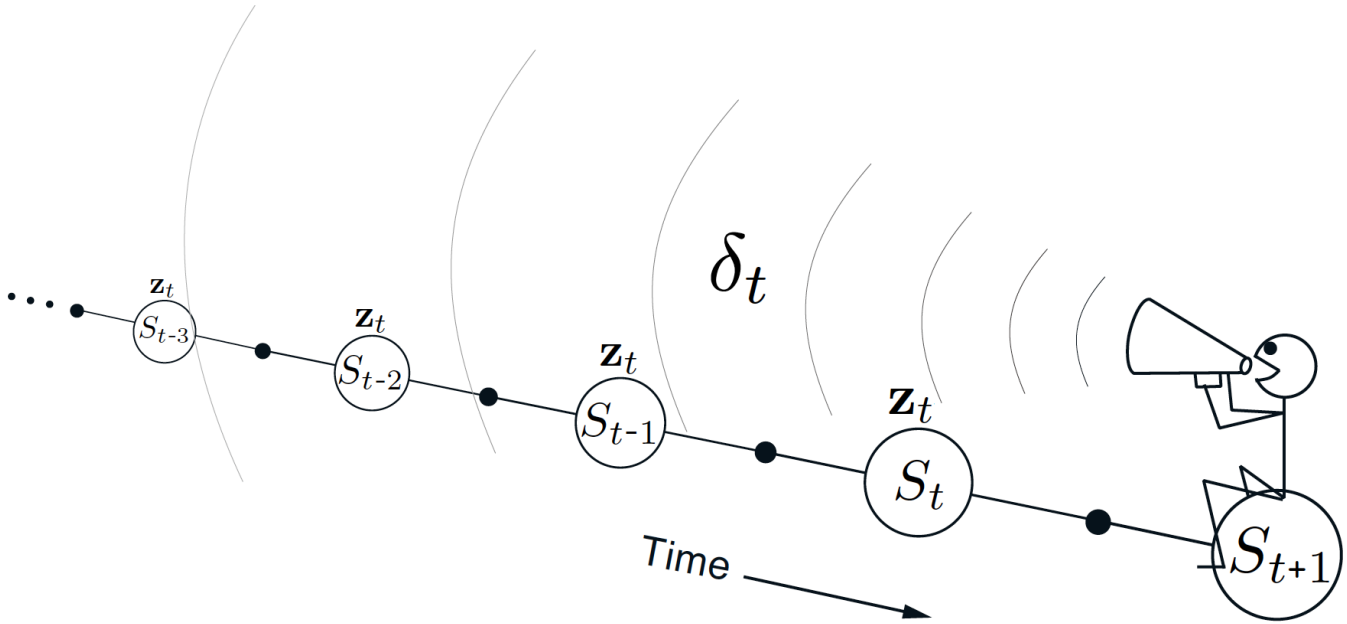


Figure 12.5: The backward or mechanistic view of TD(λ). Each update depends on the current TD error combined with the current eligibility traces of past events.

- True online TD(λ) algorithms...
- TD(λ) for control: SARSA(λ), Q(λ)

THE END