ECE7121 Learning-based control – 2025 Fall

RL basics (Classical RL)

INHA UNIVERSITY

Overview

> Tabular-value based RL (Classic RL)

> Value function

> Policy evaluation

> Policy improvement
  - policy iteration
  - value iteration

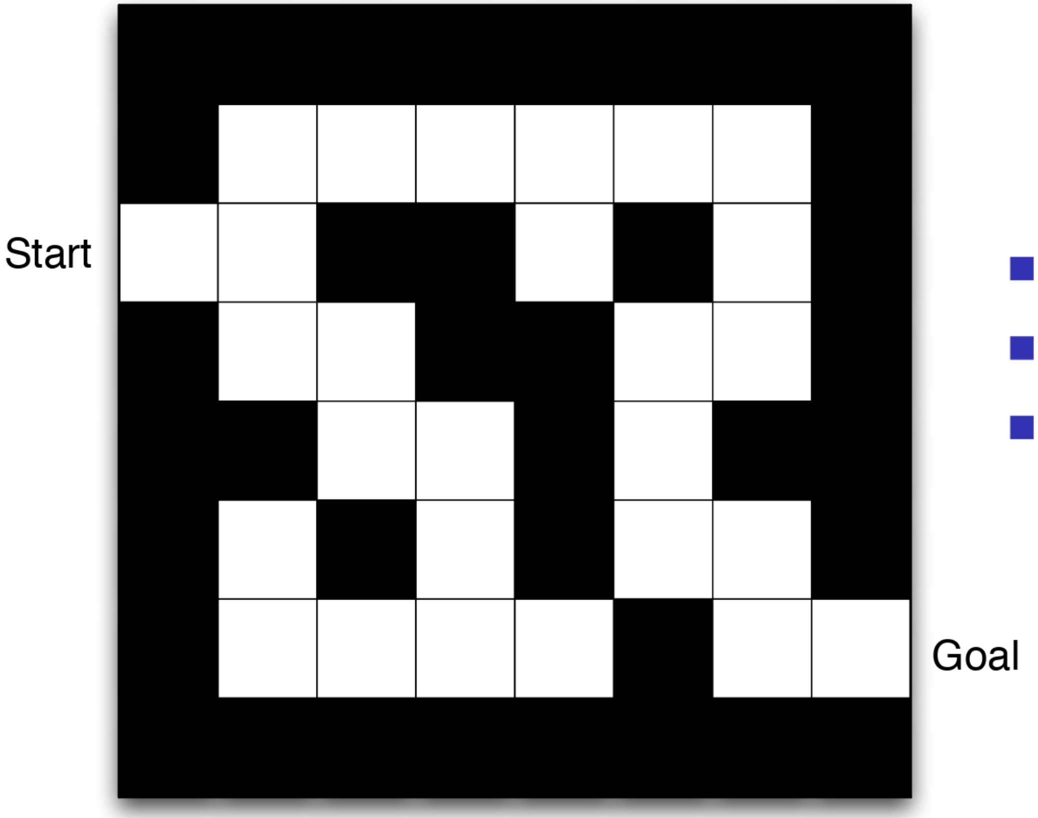
> Temporal difference learning

> Q-learning

Major components of an RL agent

- An RL agent may include those components:
  - policy: agent's behavior function
    - deterministic policy $A_t = \pi(s)$
    - stochastic policy $A_t = \pi(a|s) = p(A_t = a|S_t = s)$
  - value function: how good is each state and/or action
  - model: agent's representation of the environment
    - predict what the environments will do next
    - next state: $p(S_{t+1} = s'|S_t = s, A_t = a)$
    - next reward: $p(R_{t+1}|S_t = s, A_t = a)$

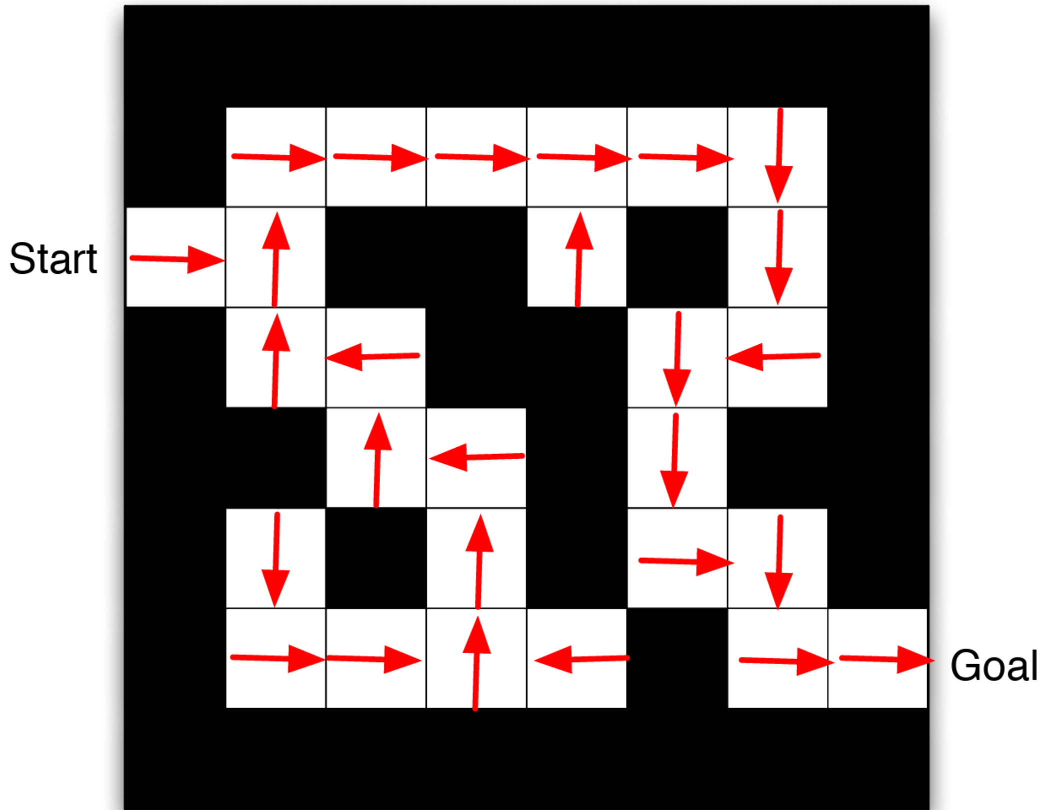
Maze example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

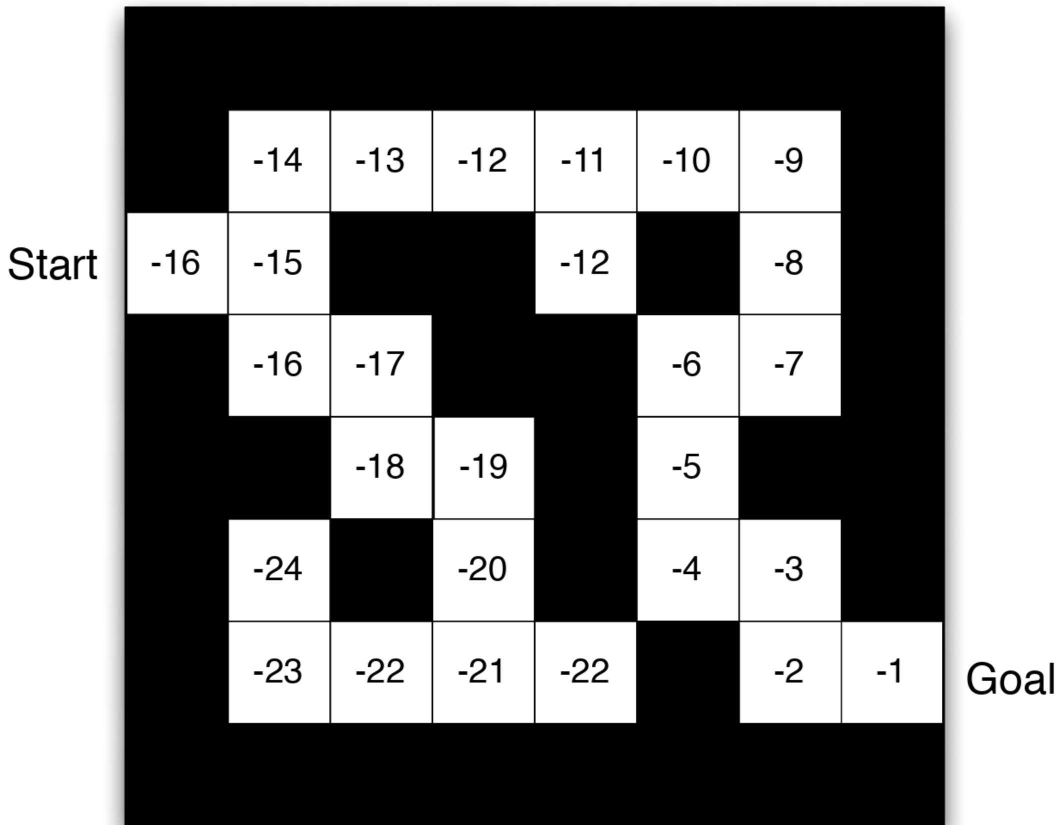
Maze example

> Policy



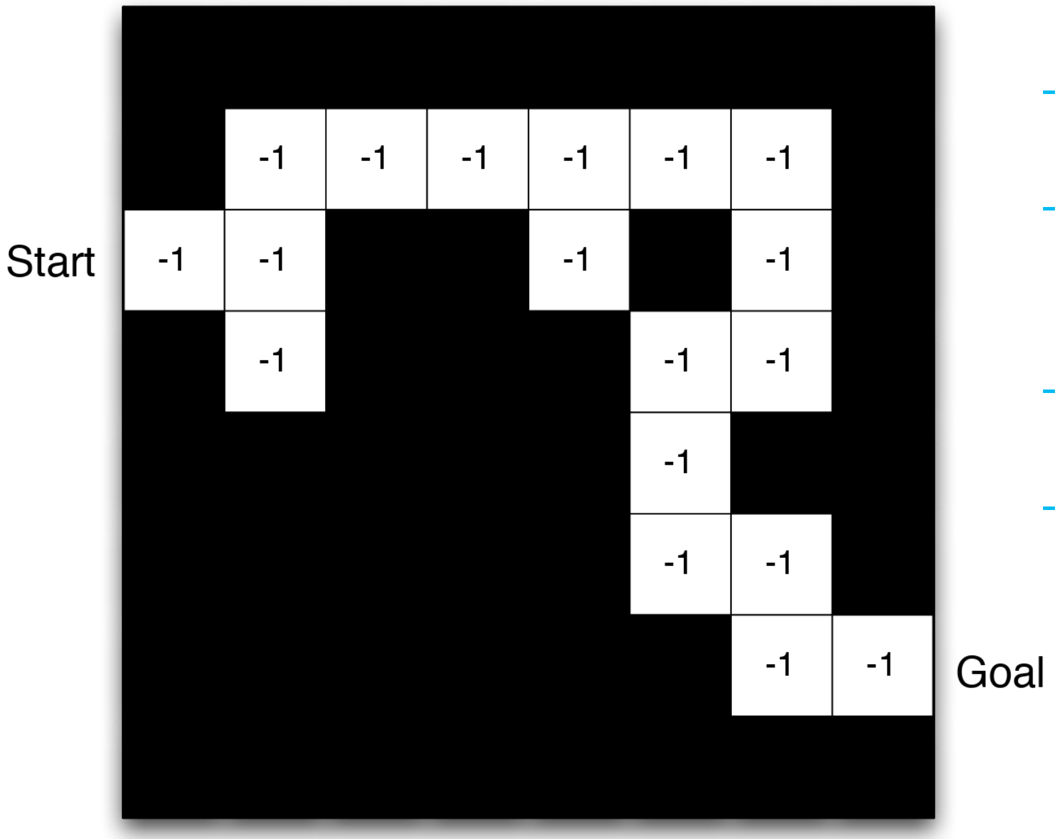
Maze example

> Value function



Maze example

> Model

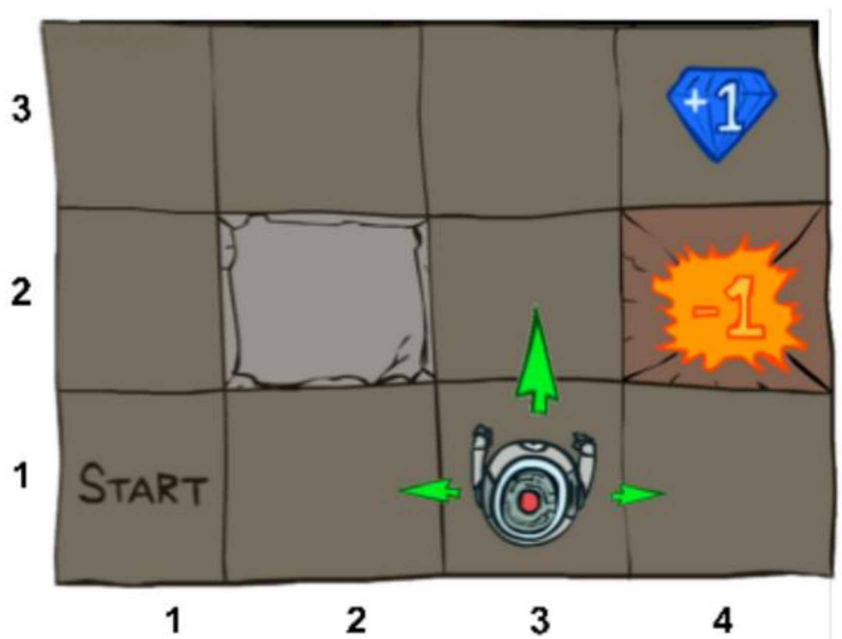


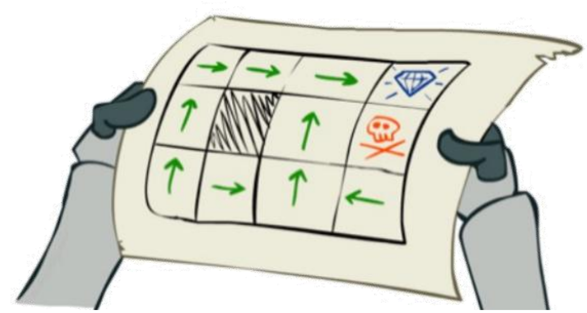
- Dynamics: how actions change the state
- Rewards: how much reward from each state

- Grid layout represents transition model
- Numbers represent immediate reward

The goal of RL

- Finite horizon case: T is finite
- Infinite horizon case: T = ∞
- The cumulative reward is often discounted: $G_t = \sum_t \gamma^t r(s_t, a_t)$
- Goal: find a policy to maximize the cumulative reward





Value functions

> State-value function (V): the expected return starting from state s, and then following policy π

- $V_\pi = \mathbb{E}_\pi[G_t | S_t = s]$
- optimal: $V^*(s) = \max_\pi V_\pi(s)$
  (maximum value function over all policies)
  (the expected return when acting optimally)

> Action-value function (Q): the expected return starting from state s, taking action a, and then following policy π

- $Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$
- optimal: $Q^*(s, a) = max_\pi \, Q_\pi(s, a)$

> Value functions capture the knowledge of the agent regarding how good is each state for the goal the agent is trying to achieve.

Value functions

> Once we obtain the optimal value function, we can find an optimal policy

> Maximizing over $Q^*(s,a)$
  - $\pi^*(a|s) = \begin{cases} 1 & if\ a = argmax_a\ Q^*(s,a) \\ 0 & otherwise \end{cases}$

> Maximizing over $V^*(s)$ with the model dynamics
  - $\pi^*(a|s) = \begin{cases} 1 & if\ a = argmax_a \sum p(s',r|s,a)(r + \gamma V^*(s')) \\ 0 & otherwise \end{cases}$
  - We need the dynamics to do one step lookahead to choose the optimal a

Roadmap

- Computing state and state-action value functions by solving linear systems of equations

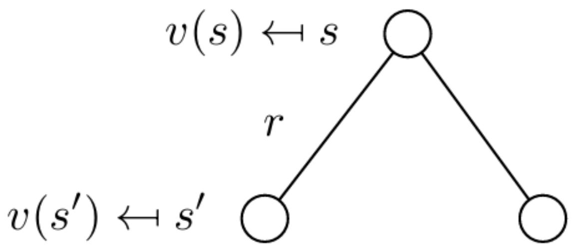
- The matrix inversion is too costly
  -> iterative estimation is required (Bellman backup operation)

- We cannot visit every state
  -> selective backups on state-actions that the agent visits

- We may not know dynamics
  -> Monte Carlo learning or TD learning

Recursive relationships for returns

> $$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$



> By taking expectations
  - $\mathbb{E}[G_t | S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s]$
  - $V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V_\pi(s') | S_t = s]$
    $= \sum_{s'} p(s', r | s)[r + \gamma V_\pi(s')]$ (Bellman expectation equation)

  - For all states, $V_\pi = R_\pi + \gamma P_\pi V_\pi$
    where v is a column vector with one entry per state

$$\begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix}$$

Solving the Bellman expectation equation

- The Bellman expectation equation is a linear equation

- It can be solved directly:
  - $V_\pi = R_\pi + \gamma P_\pi V_\pi$
    $(I - \gamma P_\pi) V_\pi = R_\pi$
    $V_\pi = (I - \gamma P_\pi)^{-1} R_\pi$

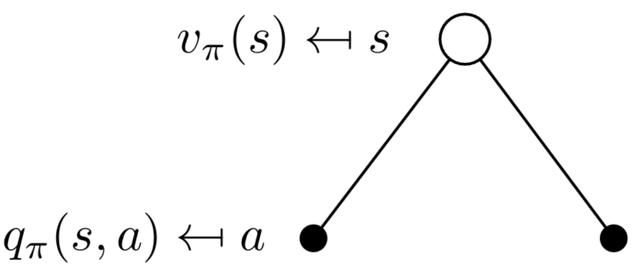
- Computational complexity is $O(n^3)$ for n states

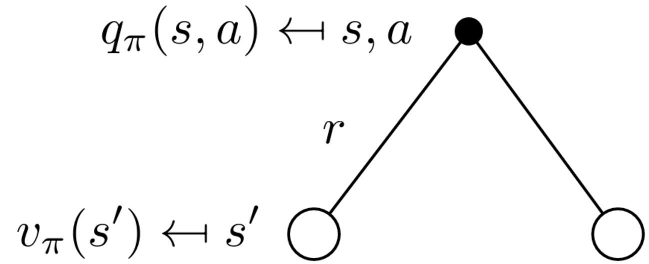
- There are many iterative methods for large state system
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Relating state and state-action value functions

> The action-value function can similarly be decomposed

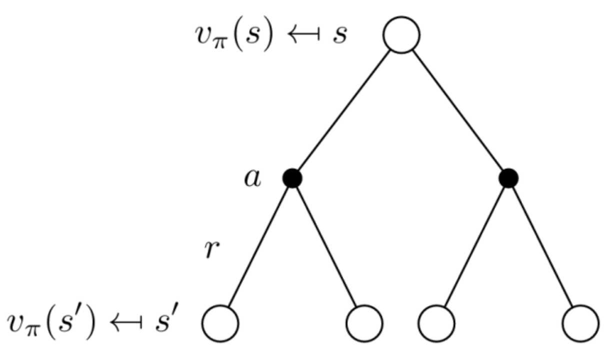
$$Q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma Q_\pi(s',a') | S_t = s, A_t = a]$$





$$V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s,a)$$

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



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

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

Bellman optimality equations

> For the Bellman expectation equations, we sum over all the possibilities.

> Now, we choose only the best action.

- $V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s,a) \quad \rightarrow \quad V^* = \max_a Q^*(s,a)$

- $Q_\pi(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma V_\pi(s')]$

  $\rightarrow Q^*(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma V^*(s')]$

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

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

- $Q_\pi(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma \sum_a \pi(a'|s') Q_\pi(s',a')]$

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

Solving the Bellman optimality equation

- Bellman optimality equation is nonlinear
- No closed form solution (in general)
- Many iterative solution methods
  - Using models / dynamic programming
    - Value iteration
    - Policy iteration
  - Using samples
    - Monte Carlo
    - Q-learning
    - SARSA

Extensions to MDPs

> Infinite and continuous MDP

> Continuous state and/or action spaces
  - closed form for linear quadratic model (LQR)

> Continuous time
  - requires partial differential equations
  - Hamilton-Jacobi-Bellman (HJB) equation

Planning by dynamic programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP

- For prediction:
  - input: MDP and policy
  - output: value function V_π

- For control:
  - input: MDP
  - output: optimal value function V^* and optimal policy π^*

Policy Evaluation

> Problem: evaluate a given policy π (prediction)

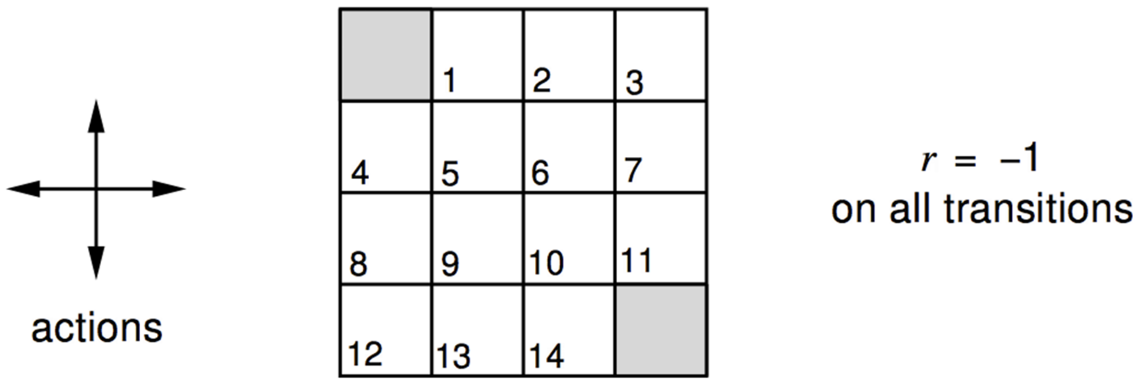
> Solution: iterative application of Bellman expectation backup
  - $V_1 \rightarrow V_2 \rightarrow \ \dots \rightarrow V_\pi$
  - Synchronous backups – for all states

> Iterative policy evaluation

> $V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} p(s', r|s, a)[r + \gamma V_k(s')]$

Policy Evaluation

> Evaluating a random policy in the small GridWorld



- undiscounted episodic MDP ($\gamma = 1$)
- one terminal state (shown twice as shaded squares)
- actions leading out of the grid leave state unchanged
- reward is -1 until the terminal state is reached
- agent follows uniform random policy
  (each action has a probability of 0.25)

Policy Evaluation

> $V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} p(s', r|s, a)[r + \gamma V_k(s')]$

v_k for the
Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Evaluation

> Overall algorithm

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    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 $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

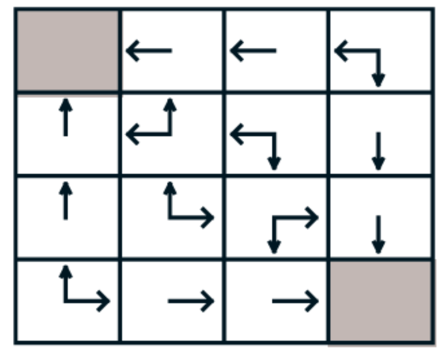
> It will converge to the fixed value function

Policy Evaluation

> Once we found the converged (optimal) value function, we can get a better (optimal) policy

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Evaluation

> Convergence proof
  - Define the Bellman expectation backup operator
  - $T_\pi(V) = R_\pi + \gamma P_\pi V$
  - This operator is a γ-contraction; it makes value functions closer by at least γ
  - $$\begin{aligned} \|T_\pi(U) - T_\pi(V)\|_\infty &= \|R_\pi + \gamma P_\pi \mathrm{U} - (R_\pi + \gamma P_\pi V)\|_\infty \\ &= \|\gamma P_\pi (U - V)\|_\infty \\ &\leq \gamma \|P_\pi\|_\infty \|U - V\|_\infty \\ &= \gamma \|U - V\|_\infty \end{aligned}$$
  - T_π converges to a unique fixed point at a linear convergence rate γ

Policy iteration

- Now, we want to move to the control problem
- Policy iteration
  - Evaluate the policy
  - Improve the policy by acting greedily with respect to V_π

$$\pi' = greedy(V_\pi)$$

  - It always converges to optimal policy π^*

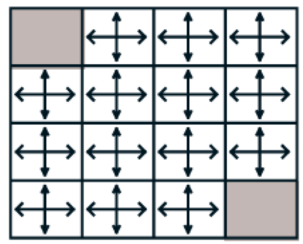
Policy iteration

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

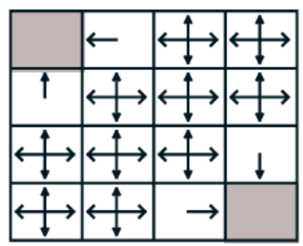
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

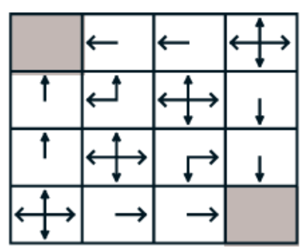
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

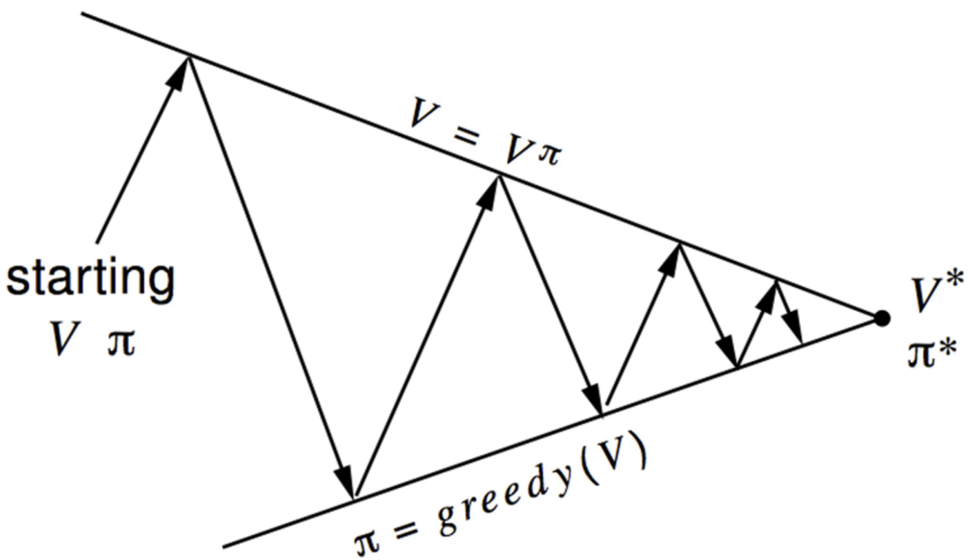
0.0	-1.0	-2.0	-2.0
-1.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.0
-2.0	-2.0	-1.0	0.0

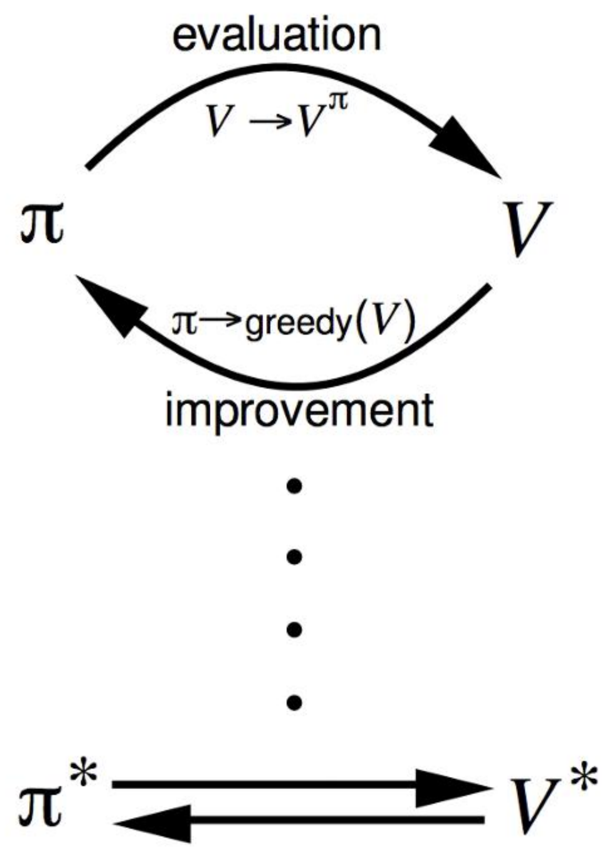


$k = 3$

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

Policy iteration





Policy iteration

> Consider a deterministic policy, $a = \pi(s)$

> Improve the policy by acting greedily, $\pi'(s) = argmax_a Q_\pi(s, a)$

> It improves the value function

- $Q_\pi\big(s, \pi'(s)\big) = \max_a Q_\pi(s, a) \geq Q_\pi\big(s, \pi(s)\big) = V_\pi(s)$
- $V_\pi(s) \leq Q_\pi\big(s, \pi'(s)\big) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma V_\pi(s')|S_t = s]$

$$\begin{aligned}
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma Q_\pi(s', \pi'(s'))|S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma V_\pi(s'')]|S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 Q_\pi(s', \pi'(s'')|S_t = s] \\
&\qquad \dots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \cdots |S_t = s] = V_{\pi'}(s)
\end{aligned}$$

Policy iteration

- If improvements stop, $Q_\pi\big(s, \pi'(s)\big) = V_\pi(s)$
- Then, the Bellman optimality equation has been satisfied

$$V^* = \max_a Q^*(s, a)$$

- Therefore, V and π are optimal

Generalized policy iteration

- Any interleaving of policy evaluation and policy improvement
- Evaluation and improvement do not need to be exact or complete at each step
  - partial updates, approximations, or asynchronous updates are allowed
- All RL methods are a form of GPI

- Policy iteration: full evaluation + full improvement
  - using Bellman expectation eqn.
- Value iteration: one-step evaluation
  - using Bellman optimality eqn.

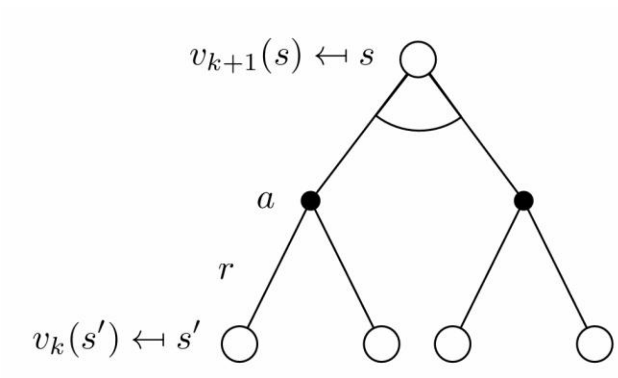
- How GPI leads optimality?
  - Over time, even with approximate steps the policy becomes progressively better

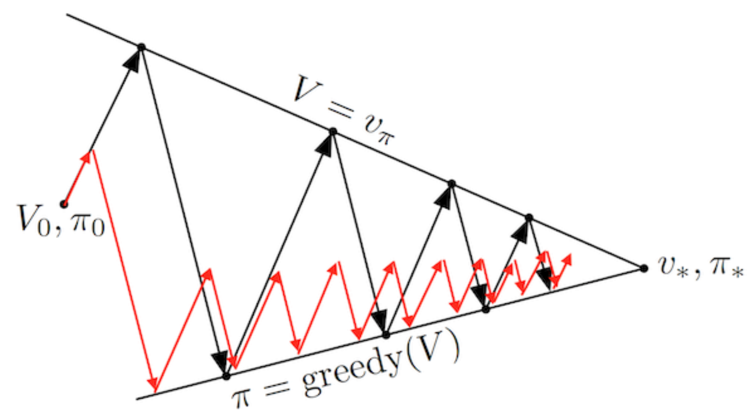
Value iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup

- Using synchronous backups
- Unlike policy iteration, there is no explicit policy

- $V_{k+1}(s) = \max_a [R(s,a) + \gamma \sum_{s'} p(s', r|s,a) V_k(s')]$





Value iteration

> $V_{k+1}(s) = \max_a [R(s,a) + \gamma \sum_{s'} p(s', r|s,a) V_k(s')]$

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Synchronous dynamic programming algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman expectation equation	Iterative policy evaluation
Control	Bellman expectation equation +Greedy policy improvement	Policy iteration
Control	Bellman optimality equation	Value iteration

- Algorithms are based on state-value function $V_\pi(s)$ or $V^*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $Q_\pi(s, a)$ or $Q^*(s, a)$
- Complexity $O(m^2n^2)$ per iteration

Asynchronous dynamic programming

- Policy/value iteration require exhaustive sweeps of the entire states.
- Asynchronous DP backs up states individually, in any order
  - Sample a state at random and apply the appropriate backup
- Can significantly reduce computation
- Guaranteed to converge if **all states continue to be selected**

Asynchronous dynamic programming

- Three simple ideas for asynchronous dynamic programming:
  - In-place dynamic programming:
    only stores one copy of value function

  - Prioritized sweeping:
    use magnitude of Bellman error to guide state selection
    backup the state with the largest remaining Bellman error
    (requires knowledge of reverse dynamics)

  - Real-time dynamic programming:
    focus on states that are relevant to agent
    (use agent's experience to guide the selection of states)

Full-width backups

- Standard DP uses full-width backups
- For each backup (sync or async)
  - every successor state and action is considered
  - using true model of transitions and reward function
- DP is effective for medium-sized problems (millions of states)
- For large problems, DP suffers from curse of dimensionality
  - even one full backup can be too expensive

Sample backups

- Sample backups: using sample rewards and sample transitions
  - Instead of reward function and transition dynamics
  - Advantages:
    - model-free: no prior knowledge of MDP required
    - breaks the curse of dimensionality through sampling
    - cost of backup is constant, independent of the state dimension

Model-free prediction and control

- Estimate or optimize the value function of an unknown MDP

- MC and TD
  - learn directly from episodes of experience
  - no knowledge of MDP transitions / rewards

- Monte-Carlo
  - learn from complete episodes: no bootstrapping
  - can only be applied to episodic MDPs (all episodes must terminate)

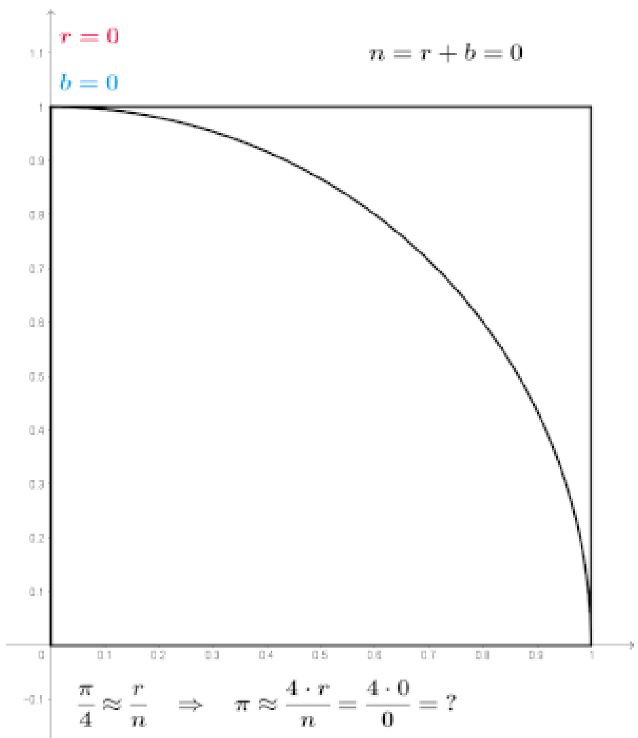
- Temporal-difference
  - learn from incomplete episodes by bootstrapping
  - update a guess towards a guess

Monte-Carlo policy evaluation

- Goal: learn V_π from episodes of experience under policy π

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return



Monte-Carlo policy evaluation

- To evaluate state s
- The first/every time-step t that state s is visited in an episode,
  - increment counter $N(s) \leftarrow N(s) + 1$
  - increment total return $S(s) \leftarrow S(s) + G_t$
  - value is estimated by mean return $V(s) = S(s)/N(s)$
  - $V(s) \rightarrow V_\pi(s)$ as $\mathrm{N(s)} \rightarrow \infty$

- Incremental updates
  - $V(s) \leftarrow V(s) + \frac{1}{N(s)}\big(G_t - V(s)\big)$
  - To forget old episodes,
    $V(s) \leftarrow V(s) + \alpha\big(G_t - V(s)\big)$

TD learning

> Incremental every-visit Monte-Carlo

- update value toward actual return G_t
- $V(s) \leftarrow V(s) + \alpha\big(G_t - V(s)\big)$

> Simplest temporal-difference learning algorithm: TD(0)

- update value towards estimated return $\mathrm{r} + \gamma V(s')$
- $V(s) \leftarrow V(s) + \alpha\big(r + \gamma V(s') - V(s)\big)$

- TD error: $\mathrm{r} + \gamma V(s') - V(s)$

TD learning

- TD can learn before knowing the final outcome
  - can learn online after every step
  - MC must wait until end of episode before return is known

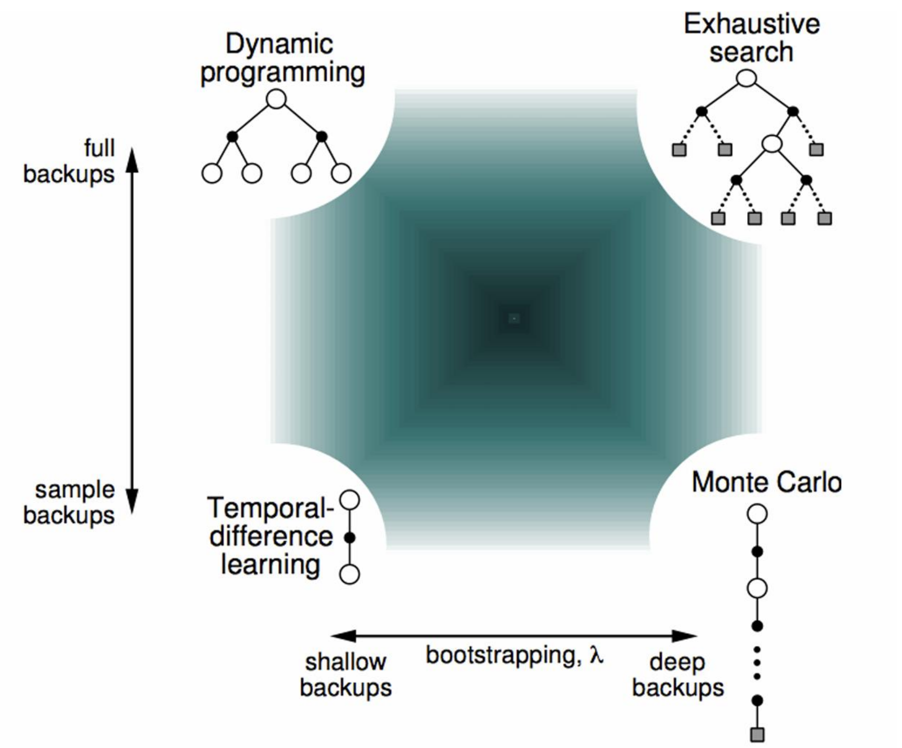
- TD can learn without the final outcome
  - can learn from incomplete sequences
  - can work in continuing (non-terminating) environments
  - while MC can't

- Disadvantage
  - Return G_t is unbiased estimate of $V(s)$, but TD target is biased
  - MC has high variance, zero bias -> good convergence
  - TD has low variance, some bias -> sensitive to initial value

Unified view of RL

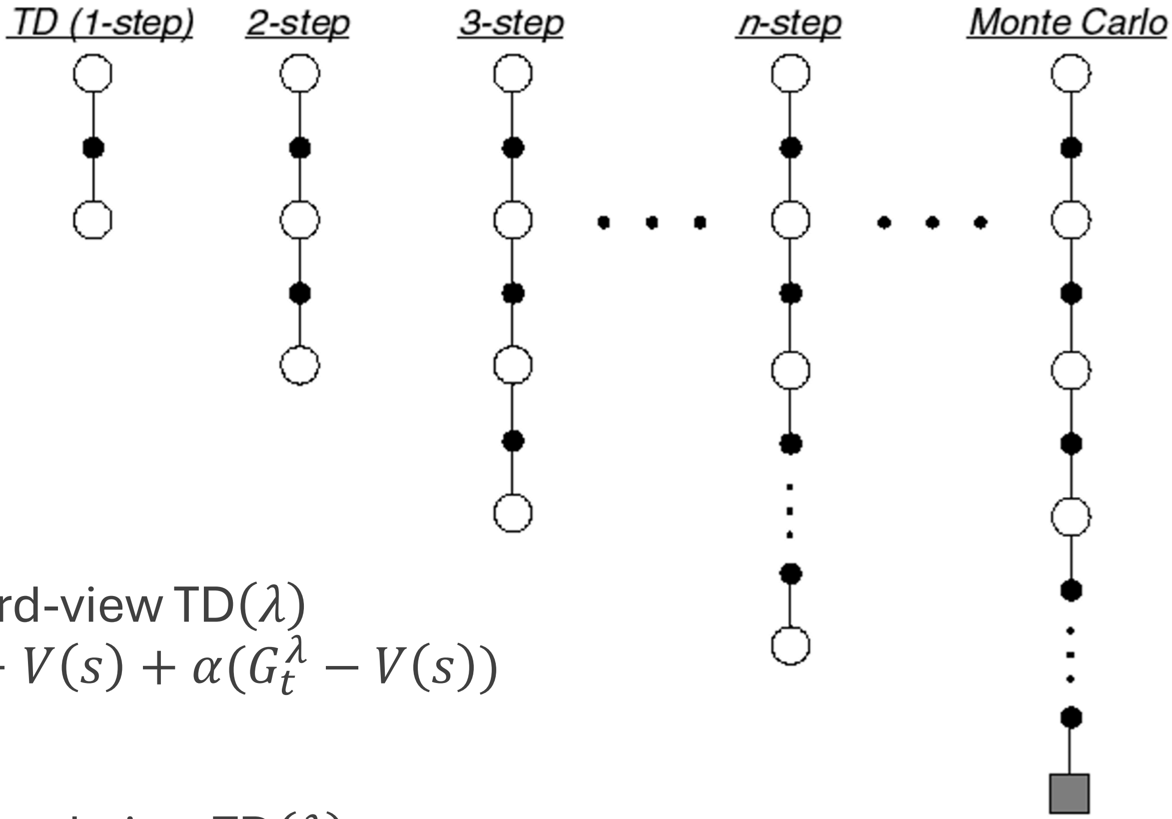
> Bootstrapping:
  update involves an estimate
  - MC does not bootstrap
  - DP bootstrap
  - TD bootstrap

> Sampling:
  update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples



TD(λ)

- n-step prediction: Let TD target look n steps into the future



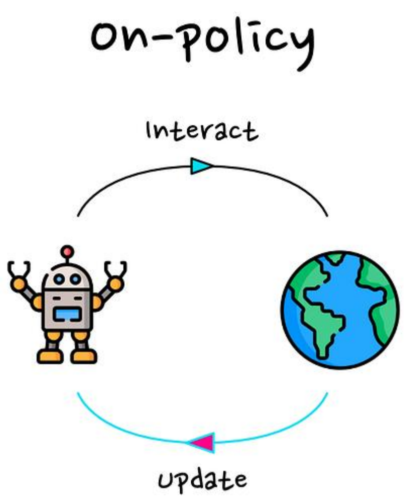
- Forward-view TD(λ)
  $V(s) \leftarrow V(s) + \alpha(G_t^\lambda - V(s))$

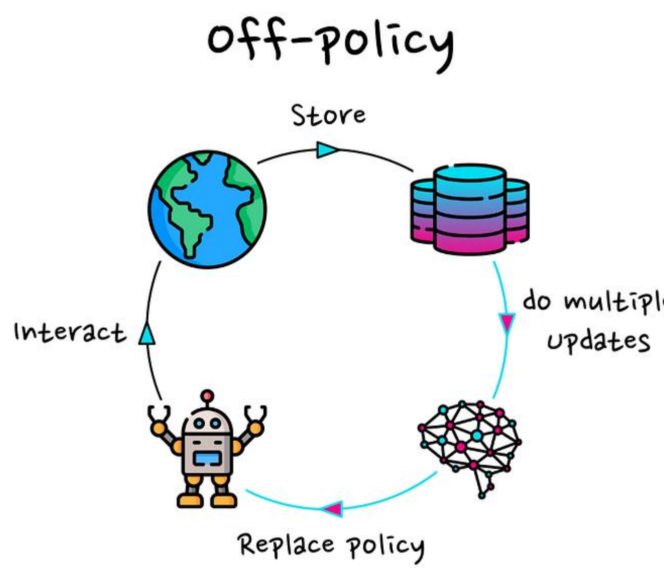
- Backward-view TD(λ)

On and off-policy learning

> On-policy learning
  - learn about policy π from experience sampled from π

> Off-policy learning
  - learn about policy π from experience sampled from other policies





On policy learning

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = argmax_a Q(s, a)$$

- Monte-Carlo policy iteration (Monte-Carlo policy eval. + ϵ-greedy)
- TD learning on action-value function: SARSA (Sarsa + ϵ-greedy)

Off policy learning

> Evaluate target policy $\pi(a|s)$ to compute $V_\pi(s)$ or $Q_\pi(s,a)$

> Importance sampling: estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X)\\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X)\\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

> Multiply importance sampling corrections along whole episode
   - can dramatically increase variance

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})}\cdots\frac{\pi(A_T|S_T)}{\mu(A_T|S_T)}G_t$$

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\pi/\mu} - V(S_t)\right)$$

Off policy learning

- Q-learning (off-policy TD learning)
  - No importance sampling is required
  - Next action is chosen using behavior policy
  - but we consider alternative successor action a^+, and update $Q(s,a)$ towards value of alternative action

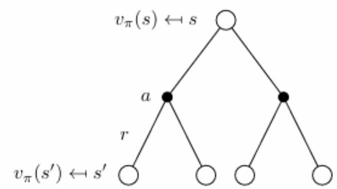
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a^+) - Q(s,a))$$

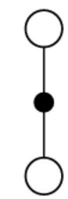
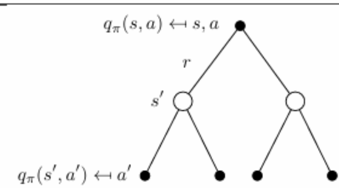
  - We allow both behavior policy and target policy to improve

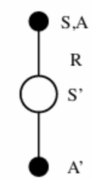
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_a Q(s',a') - Q(s,a))$$

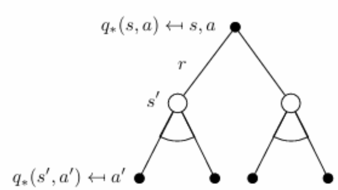
  - Q-learning control converges to the optimal action-value function

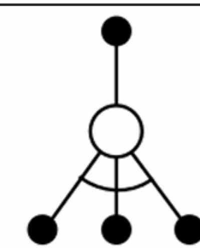
Model-free control summary

<table>
<tr><th></th><th>Full Backup (DP)</th><th>Sample Backup (TD)</th></tr>
<tr><td>Bellman Expectation Equation for $v_\pi(s)$</td><td>

Iterative Policy Evaluation</td><td>
TD Learning</td></tr>
<tr><td>Bellman Expectation Equation for $q_\pi(s, a)$</td><td>

Q-Policy Iteration</td><td>

Sarsa</td></tr>
<tr><td>Bellman Optimality Equation for $q_*(s, a)$</td><td>

Q-Value Iteration</td><td>
Q-Learning</td></tr>
</table>

Model-free control summary

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	$Q(S,A) \xleftarrow{\alpha} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \xleftarrow{\alpha} y \equiv x \leftarrow x + \alpha(y - x)$