《强化学习与控制》
--
RL Basis

Shengbo Eben Li
(李升波)

Intelligent Driving Laboratory (*i*DLab)

Tsinghua University

Blind Men and Elephant

O how they cling and wrangle, some who claim
For preacher and monk the honored name!
For, quarreling, each to his view they cling.
Such folk see only one side of a thing.

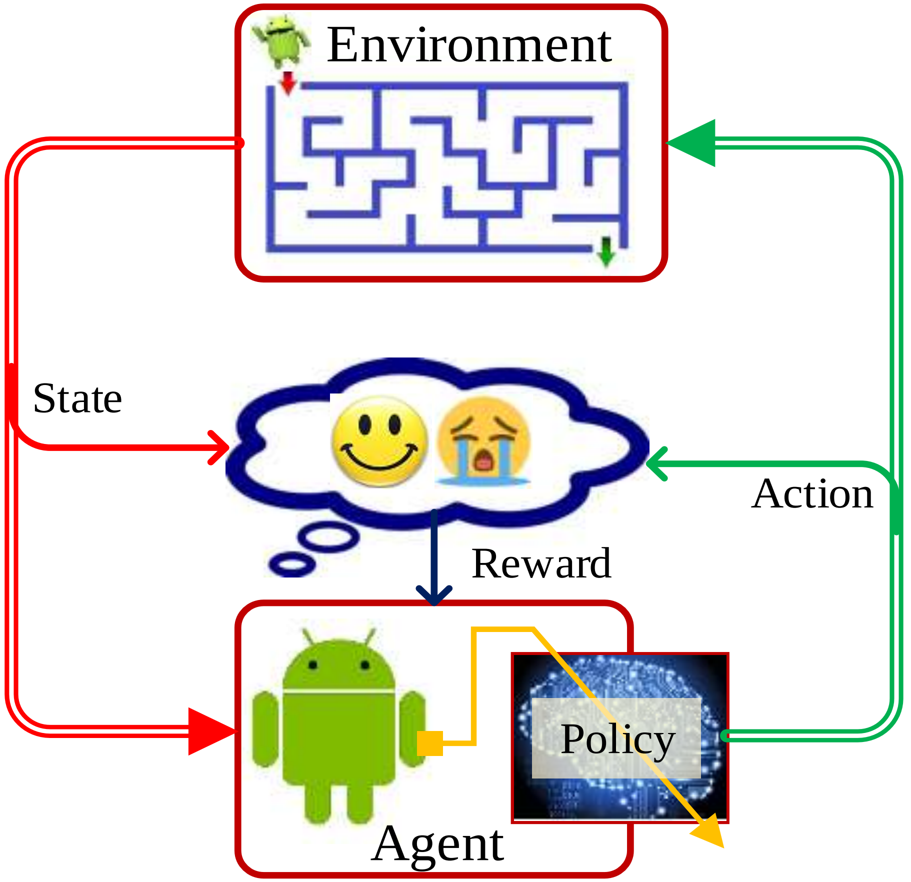
-- Gautama Buddha (563 - 483 BCE)

???
Strange! I need a clear model
Not smooth! I need regularization
It is big! I need a fast algorithm
This is a mess! I need robust algorithm

Outline

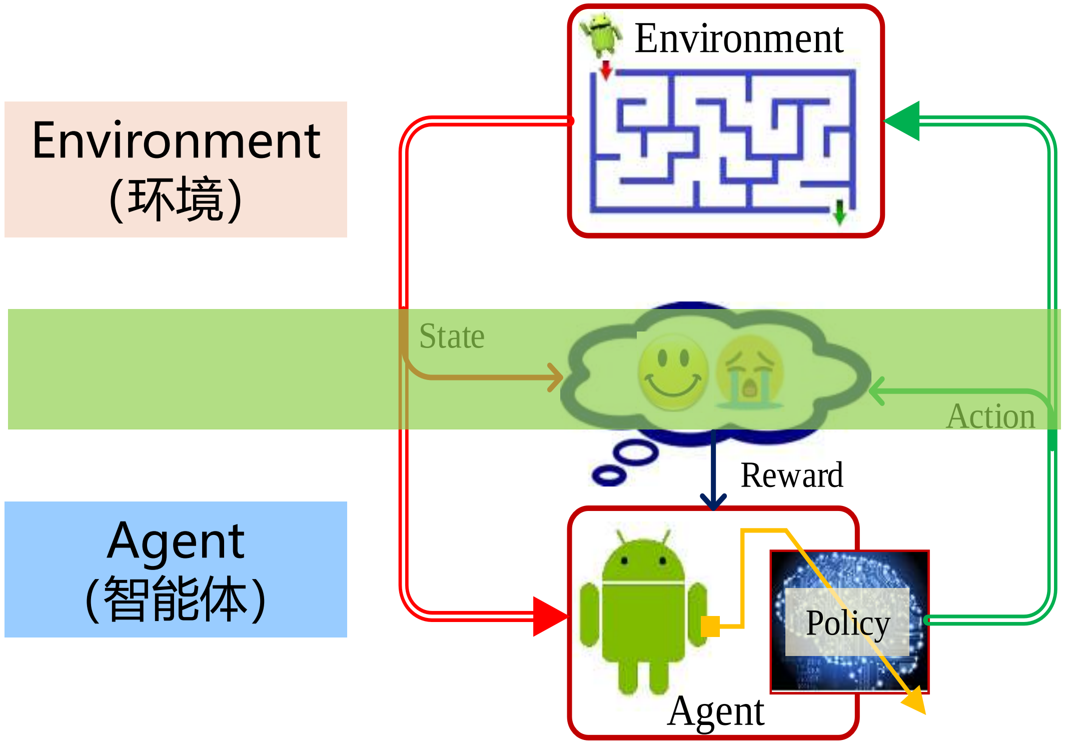
Elements of RL

- **Reinforcement Learning (RL)** **in a narrow sense**
  - Trial and error learner, inspired by animal behavior
  - Featured with repeated environment interaction



Elements of RL

- Markov Decision Process (MDP)
  - Agent（智能体）
  - Environment（环境）

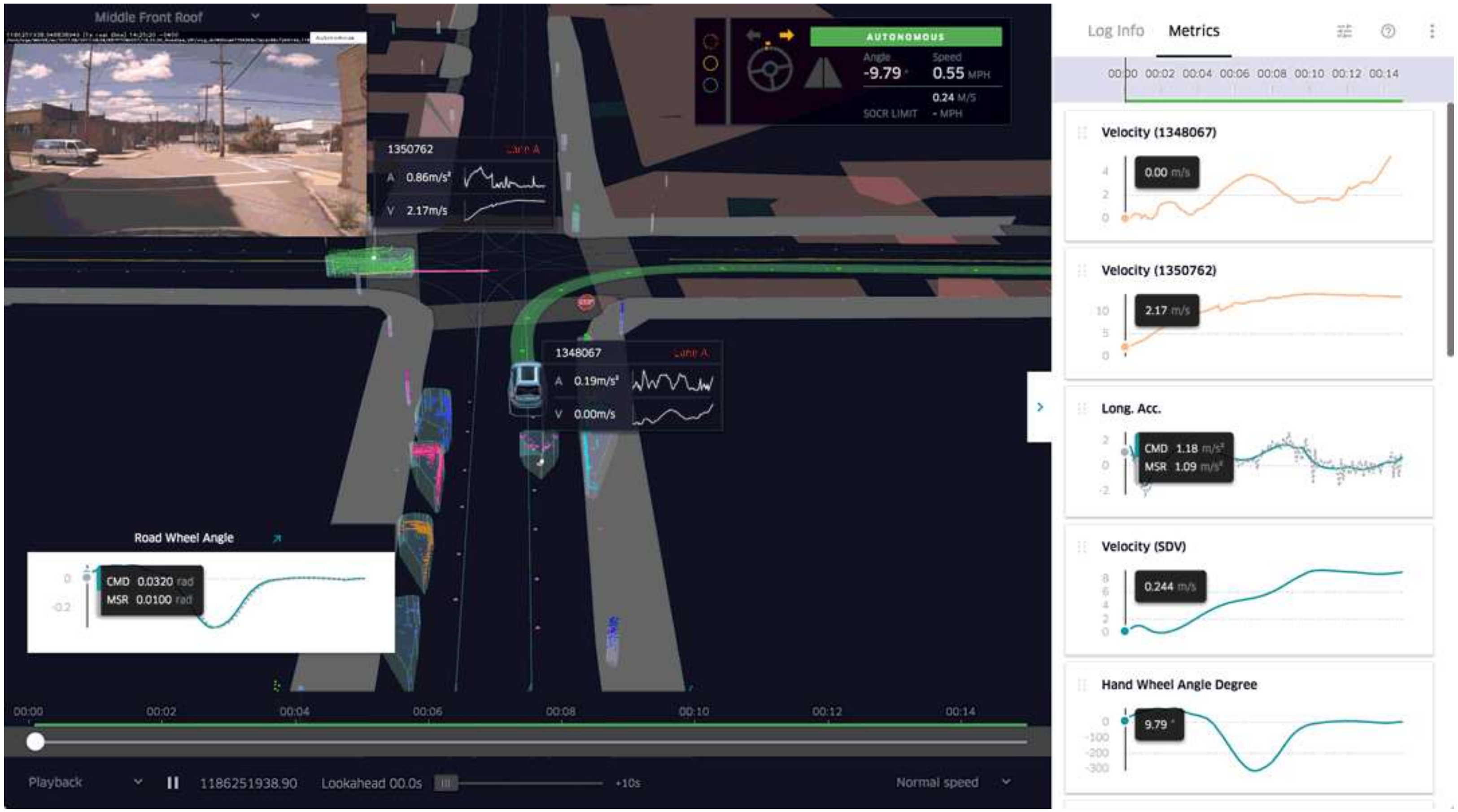


Four basic elements:

(1) Environment model
(2) State-action samples
(3) Policy
(4) Reward

Elements of RL

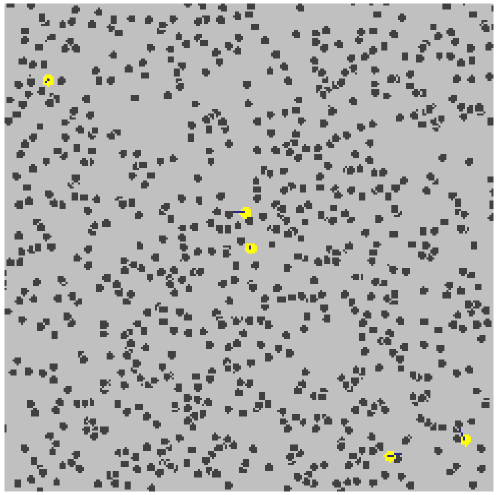
- **Agent : autonomous driving system**
- **Environment: vehicle itself, road, surrounding road users**



Environment Model

- **Markov Decision Process (MDP)**
  - Andrey Markov (Russia, 1856 – 1922)
  - Known for his work on stochastic processes
  - Markov property is named after him

	Countable state space	Continuous state space
Discrete-time	**Markov chain**	Harris chain
Continuous-time	Markov jump process	Any continuous stochastic process with Markov property, e.g., Wiener process


Brownian motion

Environment Model

Markov Property

- A state is said to have Markov property if the future is independent of the past given the present
- Conditional probability of future states depends only upon the present state, not on the history of states that preceded it

$$
\begin{aligned}
\mathcal{P}_{ss'}^{a} &\overset{\text{def}}{=} p(s'|s,a) \\
&= p(s'|s_1,a_1,s_2,a_2,\dots s_t,a_t) \\
&= \Pr\{s_{t+1}=s'|s_t=s,a_t=a\}
\end{aligned}
$$

- Current state is a sufficient representation of the environment

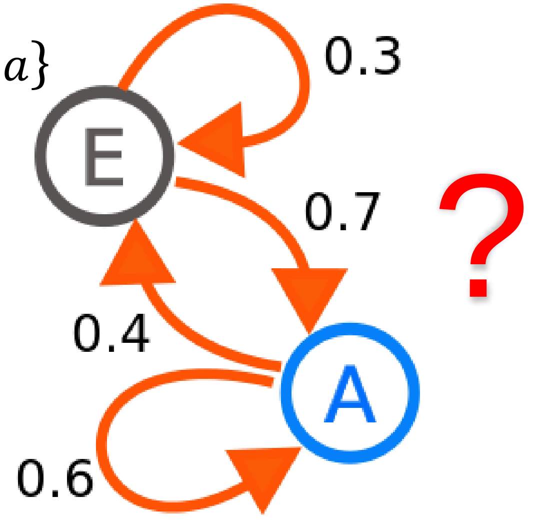
Any non-Markov random process ?

Environment Model

- **Two ways to describe stochastic environment**
  - **(1) Probabilistic model**

$$\mathcal{P}_{ss'}^{a} = p(s'|s,a) \overset{\text{def}}{=} \Pr\{s_{t+1} = s'|s_t = s, a_t = a\}$$
$$s \in \mathcal{S}, a \in \mathcal{A}$$

    - s : state
    - a : action
    - $\mathcal{S}$: a finite set of states
    - $\mathcal{A}$: a finite set of actions



  - **(2) State-action samples**
    - One sample: $< s, a, s' >$
    - One trajectory:

$$\mathcal{D} = \left\{\underbrace{s_0, a_0, s_1}_{\text{Sample}}, a_1, s_2, a_2, \cdots\cdots, \underbrace{s_t, a_t, s_{t+1}}_{\text{Sample}}, a_{t+1}, \cdots\right\}$$

Policy

- **A mapping from state space to action space that the agent has to respond**
  - Stochastic policy: probability of selecting action a at state s

$$\sum_{a \in \mathcal{A}} \pi(a|s) = 1$$

  - Deterministic policy: mapping from $s \in \mathcal{S}$ to $a \in \mathcal{A}$

$$a = \pi(s)$$

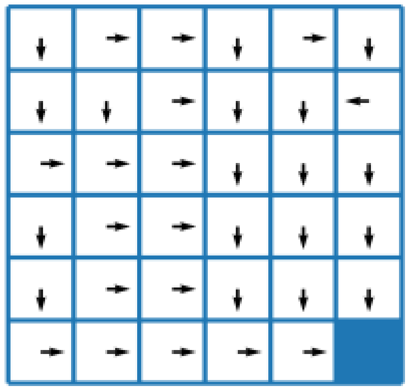
Stochastic policy in MENACE:
operate randomly for the most likely win

Deterministic policy in Robot:
action is determined by the current state

Policy

Policy representation

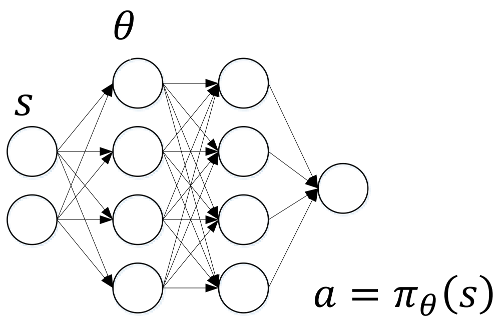
- Tabular policy

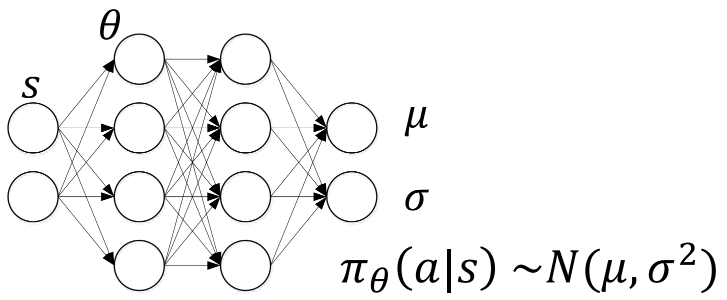


- Parameterized policy

$$\pi_\theta(s) \stackrel{\text{def}}{=} \pi(s;\theta)$$

$$\pi_\theta(a|s) \stackrel{\text{def}}{=} \pi(a|s;\theta)$$





Policy

Greedy policy

- Choose an action to maximize expected return

$$\pi^g(a|s) = \begin{cases} 1, & \text{if } a = a^* \\ 0, & \text{if } a \neq a^* \end{cases}$$

where $a^* = \arg\max_a q^\pi(s,a)$

Soft policy

- e.g., ϵ-greedy policy: behave greedily most of time, occasionally select a random action

$$\pi^\epsilon(a|s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}|}, & \text{if } a = a^* \\ \dfrac{\epsilon}{|\mathcal{A}|}, & \text{if } a \neq a^* \end{cases}$$

Reward

Immediate reward

- A function of triple (s, a, s')

$$r_{ss'}^{a} \overset{\text{def}}{=} r_t = r(s_t, a_t, s_{t+1})$$

Long-term return

- (1) Discounted return

$$G_t = \sum_{i=0}^{+\infty} \gamma^i r_{t+i}$$

  - $0 < \gamma < 1$ is discount factor

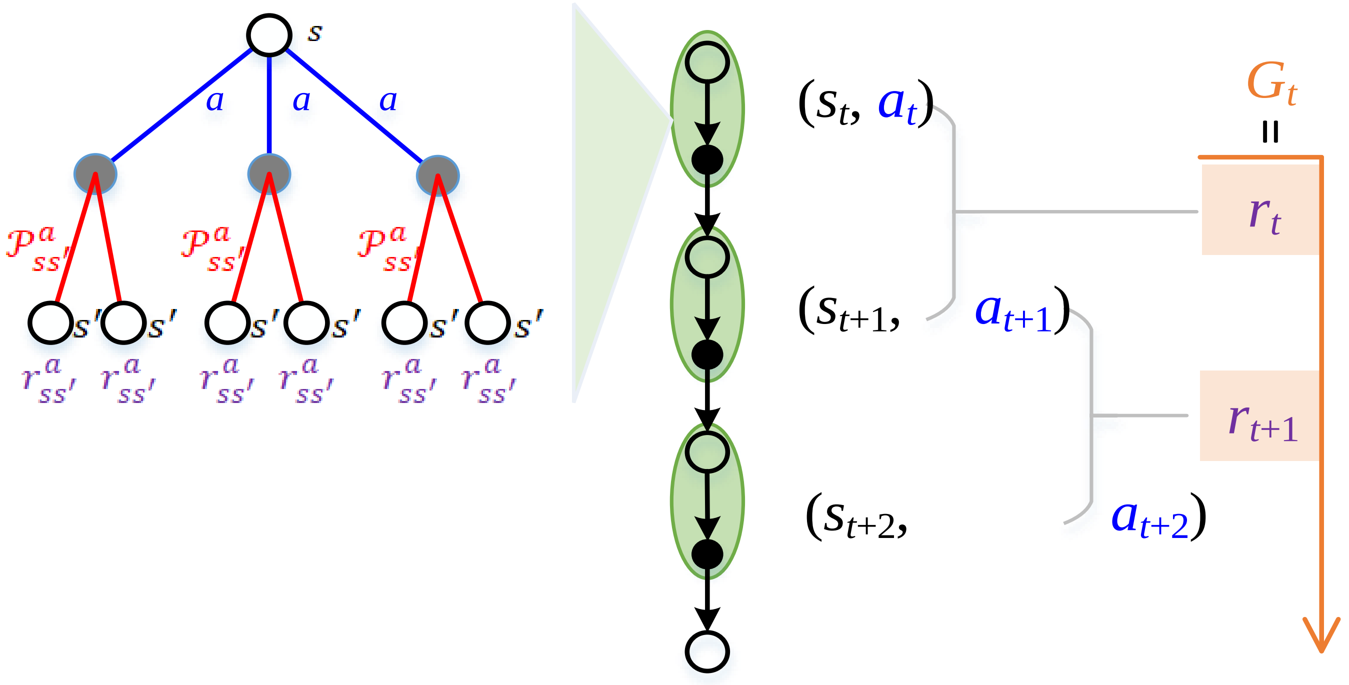
- (2) Average return

$$G_t = \lim_{T \to \infty} \frac{1}{T} \sum_{i=0}^{T-1} r_{t+i}$$

RL seeks to maximize long-term return, instead of directly optimizing immediate reward

Reward

Relationship of state, action and reward in MDPs



- Return G_t is long-term accumulated reward from time t

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{+\infty} \gamma^i r_{t+i}$$

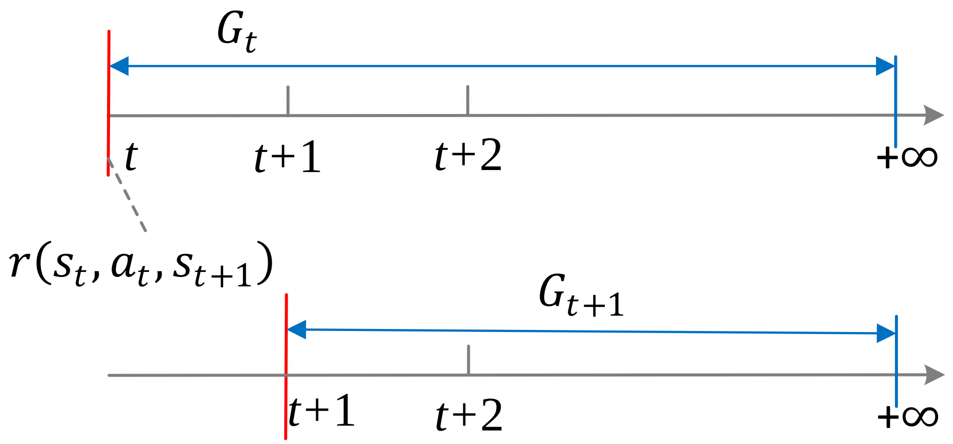
Reward

Separable Property of Long-term Return

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

$$= r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots)$$

$$= r_t + \gamma G_{t+1}$$

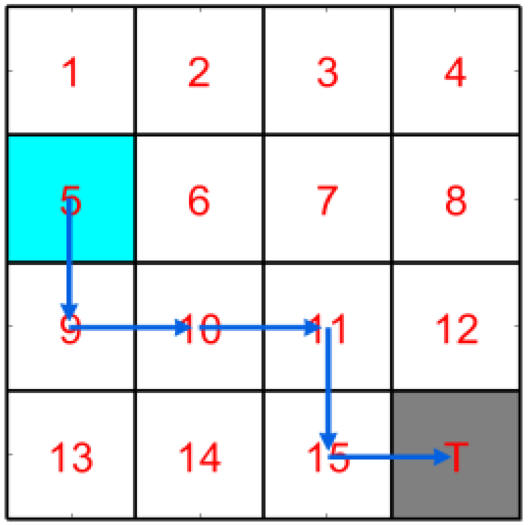


Markov property and Separable property make divide-and-conquer strategy applicable to sequential decision or optimal control

Question?

- **How to calculate long-term return?**
  - For 4×4 grid, immediate reward signal is defined:

$$r(s,a,s') = \begin{cases} -1\,, \text{if } s' \neq \mathrm{T} \\ +9\,, \text{if } s' = \mathrm{T} \end{cases}$$



Reward signals:

-1, -1,-1,-1,+9

Discount factor γ = 0.9

?

Value Function

- **Value function evaluates how good a policy behaves**
  - State-value function: expected return starting from state s under policy π

$$v^{\pi}(s) \overset{\text{def}}{=} \mathbb{E}_{\pi}\{G_t|s\} = \mathbb{E}_{\pi}\left\{\sum_{i=0}^{+\infty}\gamma^i r_{t+i}\,|s_t = s\right\}$$

$$= \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \cdots}\left\{\sum_{i=0}^{+\infty}\gamma^i r_{t+i}\,|s_t = s\right\}$$

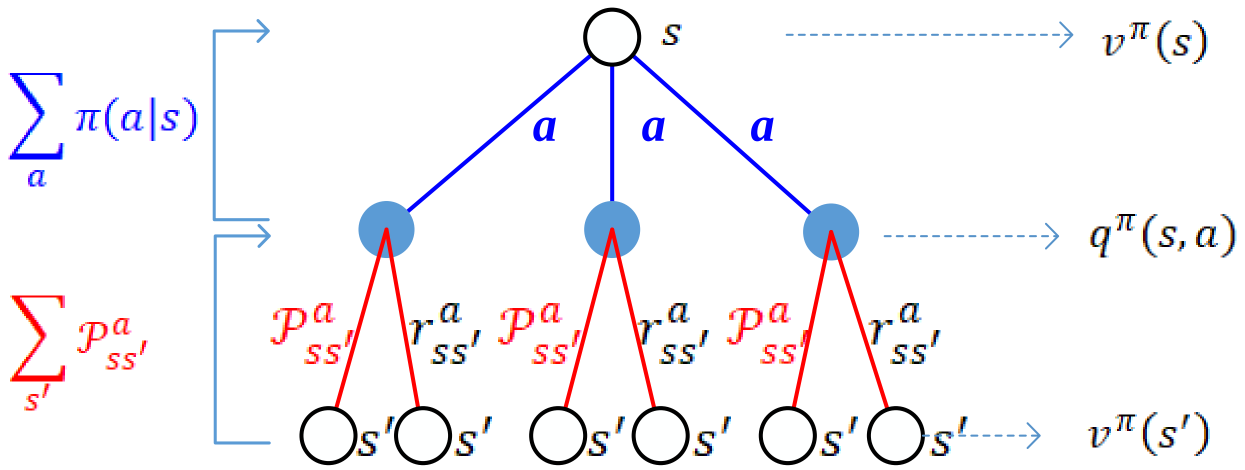
  - Action-value function: expected return under policy π when taking action a in state s

$$q^{\pi}(s,a) \overset{\text{def}}{=} \mathbb{E}_{\pi}\{G_t|s,a\} = \mathbb{E}_{\pi}\left\{\sum_{i=0}^{+\infty}\gamma^i r_{t+i}\,|s_t = s, a_t = a\right\}$$

$$= \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, \cdots}\left\{\sum_{i=0}^{+\infty}\gamma^i r_{t+i}\,|s_t = s, a_t = a\right\}$$

Relation of Two Value Functions

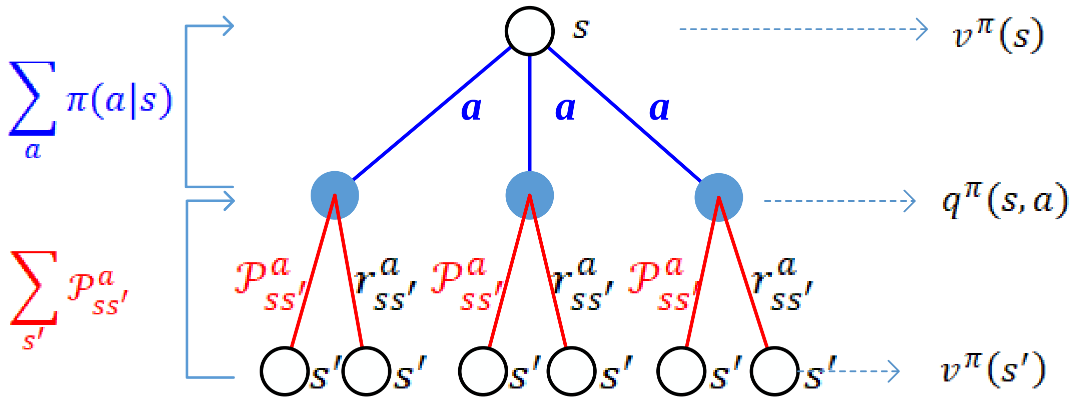
$$v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q^\pi(s,a)$$



$$q^\pi(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(r^a_{ss'} + \gamma v^\pi(s') \right)$$

Self-consistency Condition

- **Self-consistency condition of the first kind**

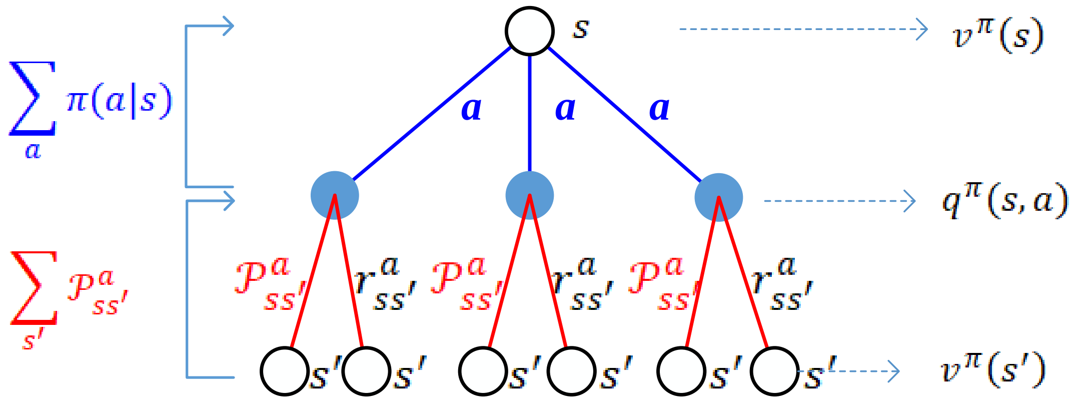


- For state-value function $v^\pi(s)$

$$v^\pi(s) = \mathbb{E}_\pi\{r + \gamma v^\pi(s')|s\}$$
$$= \sum_{a\in\mathcal{A}} \pi(a|s) \left\{ \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'} \left(r^a_{ss'} + \gamma v^\pi(s') \right) \right\}$$

Self-consistency Condition

Self-consistency condition of the second kind



- For action-value function $q^\pi(s,a)$

$$q^\pi(s,a) = \mathbb{E}_\pi\{r + \gamma v^\pi(s')|s,a\}$$

$$= \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'} \left(r^a_{ss'} + \gamma \underbrace{\sum_{a'\in\mathcal{A}} \pi(a'|s') q^\pi(s',a')}_{\boxed{v^\pi(s')}} \right)$$

Outline

Type of Tasks

Episodic task

- Chinese Go, Atari games, Gambling, etc.
- Sequences break naturally (i.e. at terminal state)， called episodes
- e.g.

$s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3, s_4 = s_T$ (Episode 1)

$s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3, s_4, a_4, r_4, s_5 = s_T$ (Episode 2)

…

$s_1, a_1, r_1, s_2, a_2, r_2, s_3 = s_T$ (Episode N)

Continuing task

- Autonomous driving, robot control, etc.
- Continuous tasks never teminate
- Episodes may be manually cut out with fixed length

Definition of RL Problems

☐ Problem definition

- To maximize the weighted expectation of state-value function while being subject to environment model or a set of collected experience

$$\max_{\pi}/\min_{\pi} J(\pi) = \mathbb{E}_{s\sim d_{\text{init}}(s)}\{v^{\pi}(s)\}$$

Initial state distribution

Subject to:

(1) $\Pr\{s_{t+1} = s' | s_t = s, a_t = a\} = \mathcal{P}_{ss'}^{a}$,

or

(2) $\mathcal{D} = \{s_0, a_0, s_1, a_1, s_2, a_2, \cdots\cdots, s_t, a_t, s_{t+1}, a_{t+1}, \cdots\}$

- Method to calculate its optimal policy
  - Indirect RL / Direct RL
- Method to use environment model
  - Model-based RL / Model-free RL

Optimal Value & Optimal Policy

Optimal value function

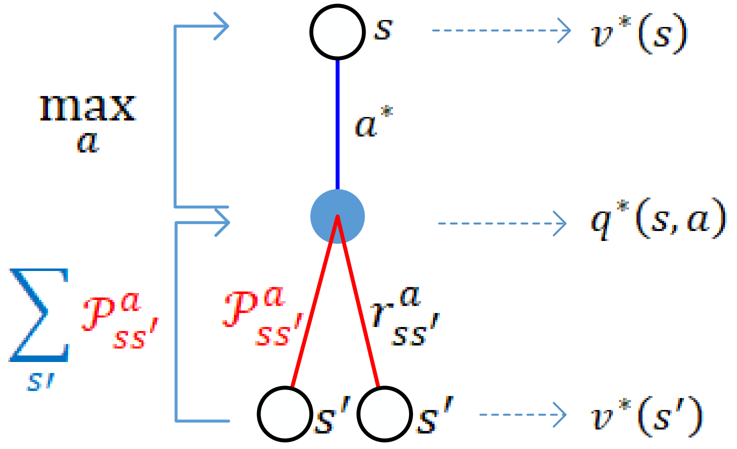
$$v^*(s) = \max_{\pi} \; v^{\pi}(s)\,, \forall s \in \mathcal{S}$$

$$q^*(s,a) = \max_{\pi} \; q^{\pi}(s,a)\,, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

- Relation:

$$v^*(s) = \max_{a \in \mathcal{A}} q^*(s,a)$$

$$q^*(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r_{ss'}^{a} + \gamma v^*(s') \right)$$



Optimal policy

- Defined as maximizing optimal action-value function

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = a^* \\ 0, & \text{if } a \neq a^* \end{cases}$$

where $a^* = \arg\max_a q^*(s,a)$

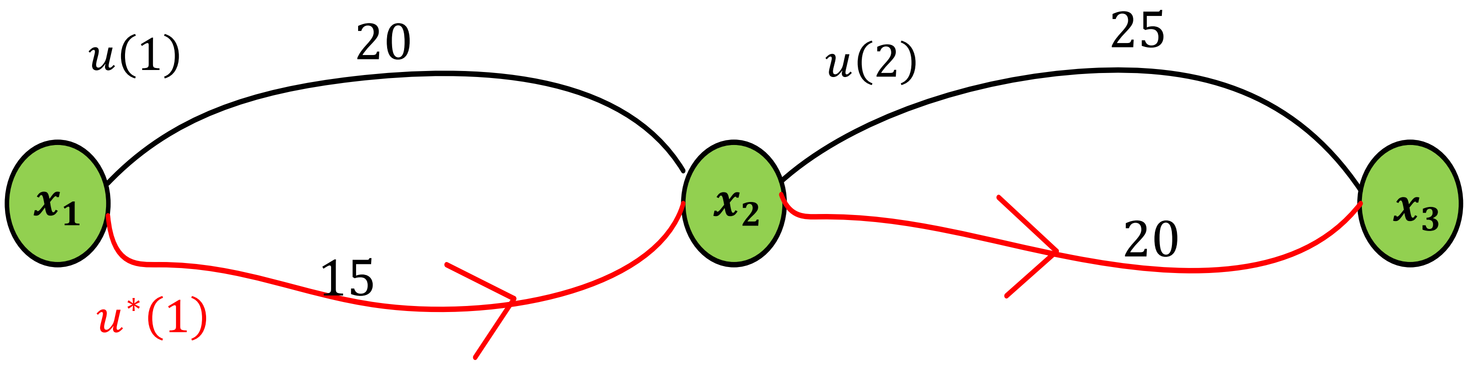
Bellman's Principle of Optimality

An optimal policy has the property that whatever the initial state and the initial decision are, the remaining decisions constitute an optimal policy with respect to the state of the system resulting from the initial decisions.

- Richard Bellman (1956)

Explanation

If x_2 belongs to an optimal path from x_1 to x_3, then any of its sub-paths $\{x_1,x_2\}$, $\{x_2,x_3\}$ is optimal, and its sub-control $u^*(1), u^*(2)$ is also optimal.



Bellman Equation

- **Bellman Equation of the first kind**

$$\begin{aligned} v^*(s) &= \max_{a\in\mathcal{A}} q^*(s,a) \\ &= \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r_{ss'}^{a} + \gamma v^*(s') \right) \end{aligned}$$

- **Bellman Equation of the second kind**

$$\begin{aligned} q^*(s,a) &= \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r_{ss'}^{a} + \gamma v^*(s') \right) \\ &= \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r_{ss'}^{a} + \gamma \max_{a'\in\mathcal{A}} q^*(s',a')) \right) \end{aligned}$$

Both of them are essentially nonlinear because of maximizers

Indirect/Direct RL

- **(1) Indirect RL**
  - Sufficient & necessary condition of optimality
    - Hamilton–Jacobi–Bellman equation (continuous-time)
    - Bellman equation (discrete-time)

$\boldsymbol{\pi}^*(\boldsymbol{a}|\boldsymbol{s})$ = Solution of HJB equation/ Bellman equation

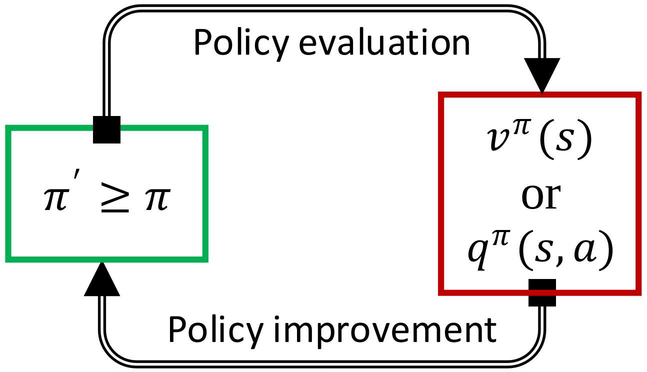
- **(2) Direct RL**
  - Search a parameterized policy that maximizes the overall RL objective function

$$\max_\theta J(\theta) \rightarrow \pi(a|s;\theta^*)$$

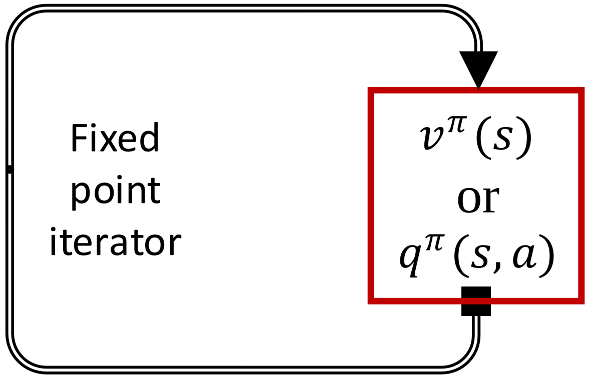
  - Iteratively update $\pi(a|s;\theta)$ to reach its optimum by using gradient descent technique

Indirect RL

- **Take the solution of Bellman equation as optimal policy**
  - (1) Policy iteration



  - (2) Value iteration

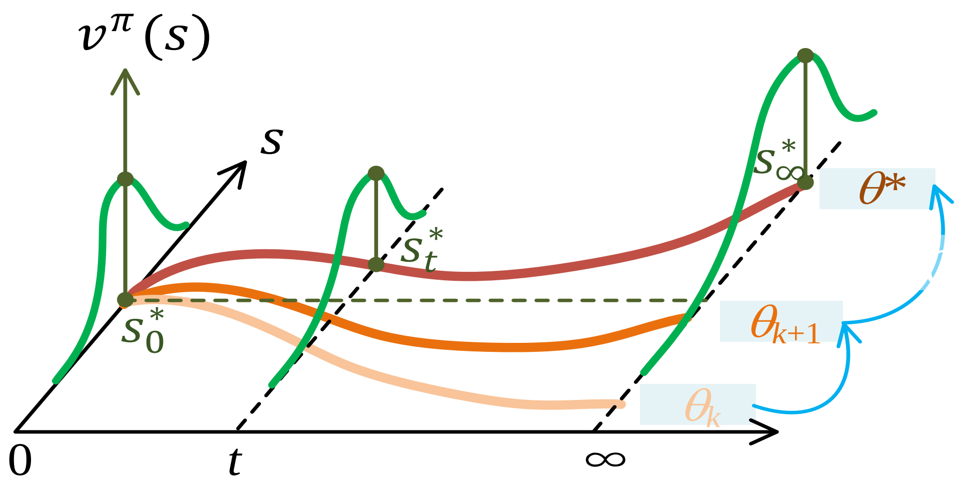


Direct RL

- **View RL as an optimization problem and calculate its optimum**
  - Search a parameterized policy with respect to the scalar performance index $J(\theta)$

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \mathbb{E}_{s\sim d_{\text{init}}(s)}\{v^{\pi_\theta}(s)\}$$

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta)$$



Summary of Today's RL Algorithms

	Model-based	Model-free
Indirect	ADP, HDP, ADHDP, DHP, GDHP, ADGDHP, CDADP, DGPI	MC, SARSA, TD, Q-learning, A2C, A3C, DQN, GAE, DDQN, Dueling DQN, C51, Rainbow, NAF, HRA, HER
Direct	PILCO, GPS, I2A, MVE, STEVE, PETS, MBMF, AMRL	TRPO, PPO, DPG, DDPG, Off-PAC, ACER, REACTOR, IPG, TD3, SAC, DSAC, Trust-PCL, SIL, APE-X, IMPALA

- ADP: Approximate Dynamic Programming; HDP: Heuristic Dynamic Programming; ADHDP: Action Dependent Heuristic Dynamic Programming; DHP: Dual Heuristic Dynamic Programming; GDHP: Globalized Dual Heuristic Dynamic Programming; ADGDHP: Action Dependent Globalized Dual Heuristic Dynamic Programming.
- MC: Monte Carlo; Sarsa: State-Action-Reward-State-Action; Q-learning: Quality-oriented learning; A2C: Advantage Actor-Critic; A3C: Asynchronous Advantage Actor-Critic; DQN: Deep Q Networks; GAE: Generalized Advantage Estimator; DDQN: Double Deep Q Networks; Dueling DQN: Dueling Deep Q Networks; C51: Distributional Perspective on Reinforcement Learning; Rainbow: Combining Improvements in Deep Reinforcement Learning; NAF: Normalized Advantage Functions.
- PILCO: Model-Based and Data-Efficient Approach to Policy Search; GPS: Guided Policy Search; I2A: Imagination-Augmented Agents; MVE: Model-based Value Expansion; STEVE: Stochastic Ensemble Value Expansion.
- TRPO: Trust Region Policy Optimization; PPO: Proximal Policy Optimization; DPG: Deterministic Policy Gradient; DDPG: Deep Deterministic Policy Gradient; Off-PAC: Off-policy Actor-Critic; ACER: Actor-Critic with Experience Replay; REACTOR: Distributional Retrace Actor-Critic; IPG: Interpolated Policy Gradient; TD3: Twin Delayed Deep Deterministic Policy Gradient; SAC: Soft Actor-Critic; DSAC: Distributional Soft Actor-Critic; Trust-PCL: Trust Path Consistency Learning; SIL: Self-Imitation Learning; APE-X: Distributed Prioritized Experience Replay; IMPALA: Importance Weighted Actor-Learner Architectures

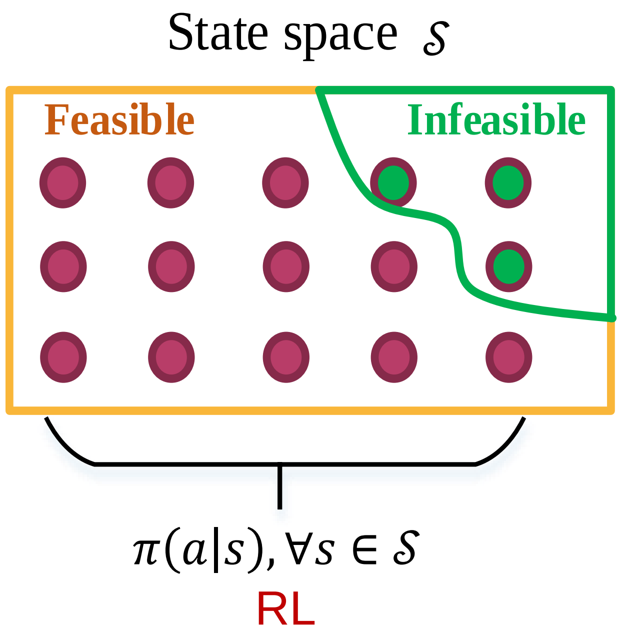
Outline

Problem Definition: RL vs MPC

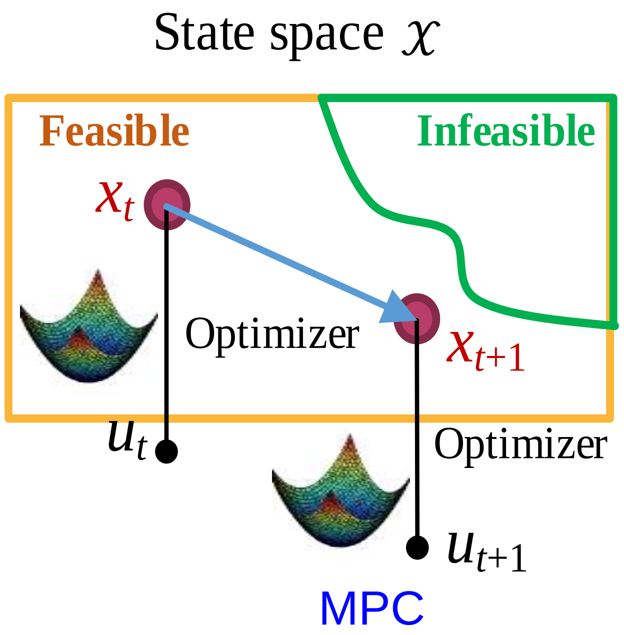
RL w/ Discounted cost)	MPC w/ Infinite horizon
Stochastic system (e.g., environment)	Deterministic system (e.g., plant, process)
State & Action (s, a)	State & Control input (x, u)
Probabilistic model $\Pr\{s_{t+1} = s' \| s_t = s, a_t = a\} = \mathcal{P}_{ss'}^{a}$	State-space model $x_{t+1} = f(x_t, u_t)$
Policy $\pi(a\|s)$	Controller $u = \pi(x)$
Reward signal $r(s, a, s')$	Utility function $l(x, u)$
State-value function $v^{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{i=0}^{+\infty} \gamma^i r_{t+i} \,\|\, s_t = s\right\}$	Infinite-horizon cost function $V(x) = \sum_{i=0}^{+\infty} l(x_{t+i}, u_{t+i})\Big\|_{x_t = x}$
Maximize a weighted state-value function $\max_{\pi} \mathbb{E}_{s\sim d(s)}\{v^{\pi}(s)\}$	Minimize a cost function $\min_{u} V(x)$
Self-consistency condition $v^{\pi}(s) = \sum_{a} \pi(a\|s)\left\{\sum_{s'} \mathcal{P}\left(r + \gamma v^{\pi}(s')\right)\right\}$	Self-consistency condition $V(x) = l(x, u) + V(x')$
Bellman equation $v^*(s) = \max_{a} \sum_{s'} \mathcal{P}\left(r + \gamma v^*(s')\right)$	Bellman equation $V^*(x) = \min_{u}\{l(x, u) + V^*(x')\}$

How to Find Optimum: RL vs MPC

- **RL: find optimal policy for the whole state space**
- **MPC: find an optimal action for each state point**



State space $\mathcal{S}$
Feasible
Infeasible
$\pi(a|s), \forall s \in \mathcal{S}$
RL



- **RL**: prone to fail even though only some local states have no allowable action
- **MPC:** better tolerance to infeasibility since it does not enter the unallowable region

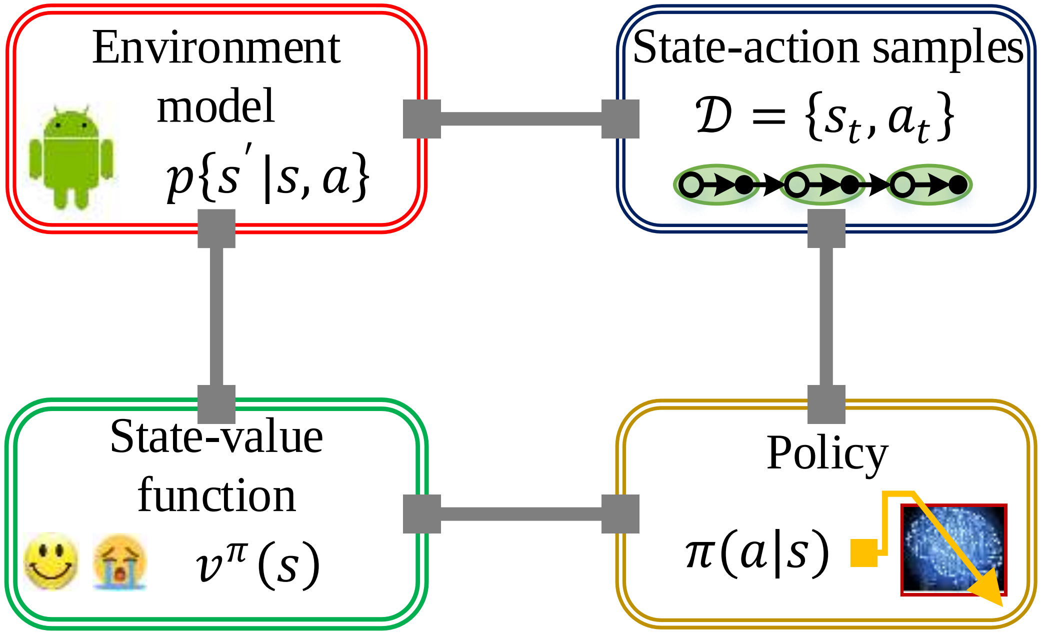
Combination of Four Elements

Revisit four basic elements

- State-action samples

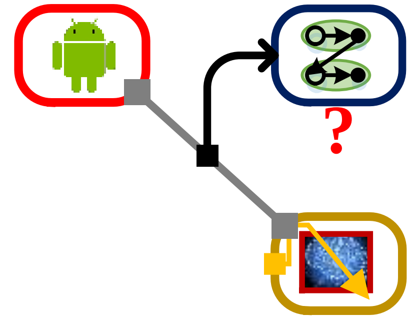
  $$\mathcal{D} = \{s_0, a_0, s_1, a_1, s_2, a_2, \cdots\cdots, s_t, a_t, s_{t+1}, a_{t+1}, \cdots\}$$

- Policy $\pi(a|s)$
- State-value function $v^\pi(s)$
- Environment model $p\{s'|s,a\}$

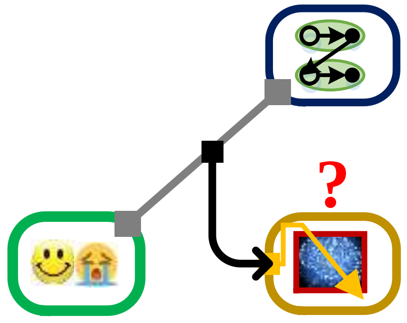


Combination of Four Elements

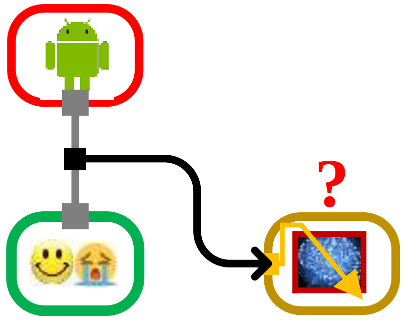
- **Different kinds of problem definition**



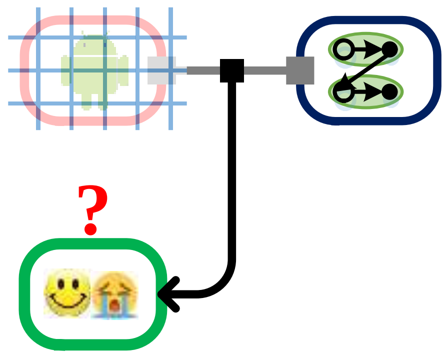
(a) Simulate a system



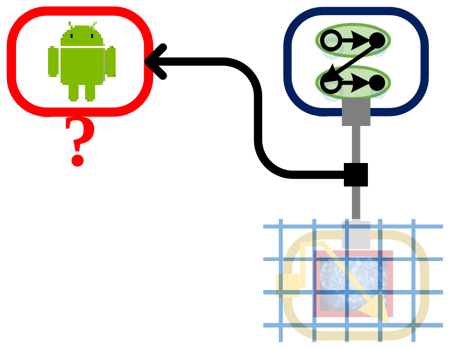
(b) Model-free RL



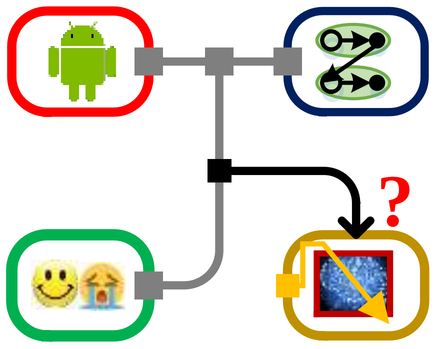
(c) Model-based RL



(d) Inverse RL



(e) Parameter identification



(f) Mixed RL

Outline

Measures of Learning Performance

- **(1) Policy performance**
  - Total average return (TAR)

$$R_{\text{Avg}} = \sum d_{\text{init}}(s) \left\{ \frac{1}{N} \sum\nolimits_{i=1}^{N} G_i(s) \right\}$$

  - Different from value function, each return $G_i(s)$ should be calculated without discounting information (for fair comparison)

- **(2) Learning accuracy**
  - Root mean square error (RMSE)

$$\text{RMSE}_{v\text{Opt}} = \sqrt{\sum_{s \in \mathcal{S}} d_\pi(s) \left(V^\pi(s) - v^*(s) \right)^2}$$

$$\text{RMSE}_{\pi\text{Opt}} = \sqrt{\sum_{s \in \mathcal{S}} d_\pi(s) \left\| \pi(s) - \pi^*(s) \right\|_2^2}$$

Measures of Learning Performance

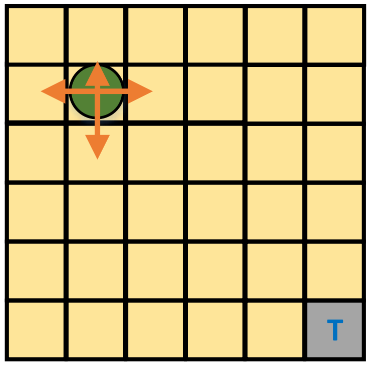
- **(3) Learning speed**
  - Wall-clock time (depend on computer performance)
  - Iteration efficiency
- **(4) Sample efficiency**
  - Collected samples that are required to reach a certain performance
  - Re-used samples are not counted, i.e., experience replay
- **(5) Approximate accuracy**
  - Average RMSE (Root Mean Square Error)

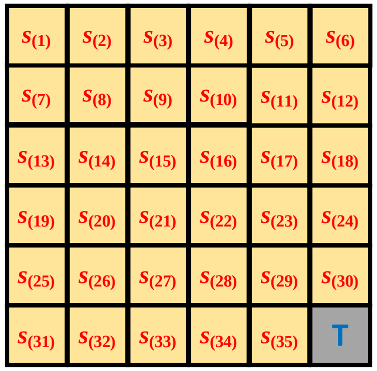
$$\mathrm{RMSE}_{v\mathrm{Appr}} = \sqrt{\sum_{s \in \mathcal{S}} d_{\pi}(s) \left(V^{\pi}(s;w) - v^{\pi}(s) \right)^2}$$

    - $V^{\pi}(s;w) > v^{\pi}(s)$: overestimation
    - $V^{\pi}(s;w) < v^{\pi}(s)$: underestimation

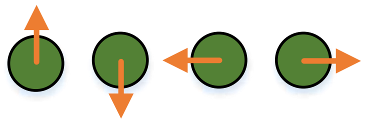
Example: Indoor cleaning robot

- **Indoor cleaning robot**
  - Assist humans in floor cleaning




$s_{(1)}$ $s_{(2)}$ $s_{(3)}$ $s_{(4)}$ $s_{(5)}$ $s_{(6)}$
$s_{(7)}$ $s_{(8)}$ $s_{(9)}$ $s_{(10)}$ $s_{(11)}$ $s_{(12)}$
$s_{(13)}$ $s_{(14)}$ $s_{(15)}$ $s_{(16)}$ $s_{(17)}$ $s_{(18)}$
$s_{(19)}$ $s_{(20)}$ $s_{(21)}$ $s_{(22)}$ $s_{(23)}$ $s_{(24)}$
$s_{(25)}$ $s_{(26)}$ $s_{(27)}$ $s_{(28)}$ $s_{(29)}$ $s_{(30)}$
$s_{(31)}$ $s_{(32)}$ $s_{(33)}$ $s_{(34)}$ $s_{(35)}$ T

State = $\{s_{(i)}\}$, i=1, 2,… , 35, 36



Action = {North, South, West, East}

Grid environment

State space

Action space

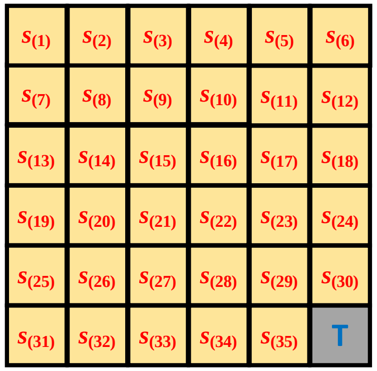
Example: Indoor cleaning robot

- **Environment model**

$$\Pr\{s' = \text{Front Cell } | s, a\} = 0.8$$
$$\Pr\{s' = \text{Left Cell } | s, a\} = 0.1$$
$$\Pr\{s' = \text{Right Cell } | s, a\} = 0.1$$
$$\Pr\{s' = \text{Back Cell } | s, a\} = 0$$

- **Reward**

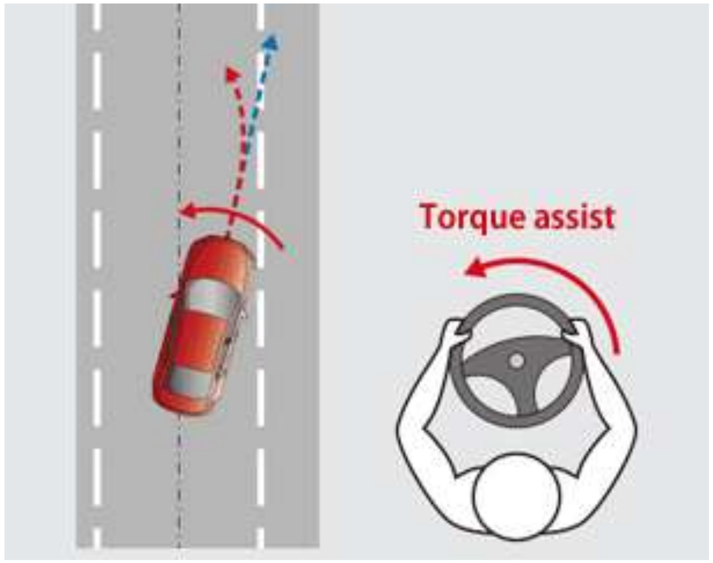
$$r(s, a, s') = \begin{cases} -1 & \text{if } s' \neq \mathrm{T} \\ +9, & \text{if } s' = \mathrm{T} \end{cases}$$

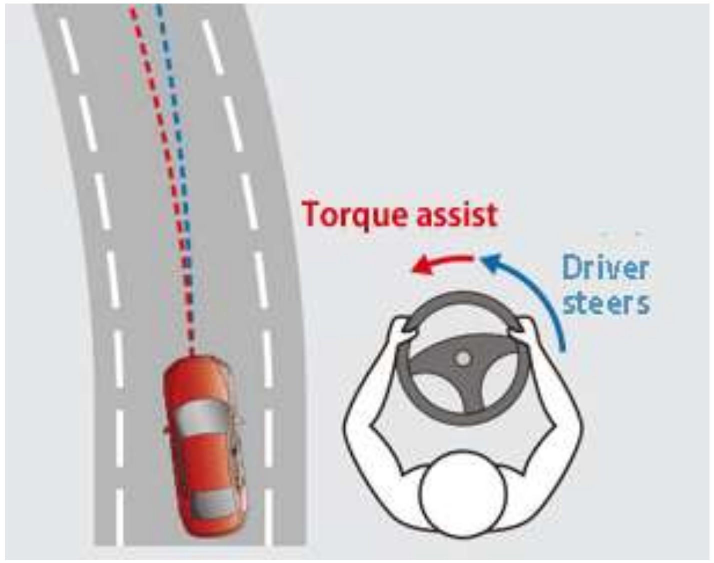


$s_{(1)}$ $s_{(2)}$ $s_{(3)}$ $s_{(4)}$ $s_{(5)}$ $s_{(6)}$
$s_{(7)}$ $s_{(8)}$ $s_{(9)}$ $s_{(10)}$ $s_{(11)}$ $s_{(12)}$
$s_{(13)}$ $s_{(14)}$ $s_{(15)}$ $s_{(16)}$ $s_{(17)}$ $s_{(18)}$
$s_{(19)}$ $s_{(20)}$ $s_{(21)}$ $s_{(22)}$ $s_{(23)}$ $s_{(24)}$
$s_{(25)}$ $s_{(26)}$ $s_{(27)}$ $s_{(28)}$ $s_{(29)}$ $s_{(30)}$
$s_{(31)}$ $s_{(32)}$ $s_{(33)}$ $s_{(34)}$ $s_{(35)}$ T

State = $\{s_{(i)}\}$, i=1, 2,… , 35, 36

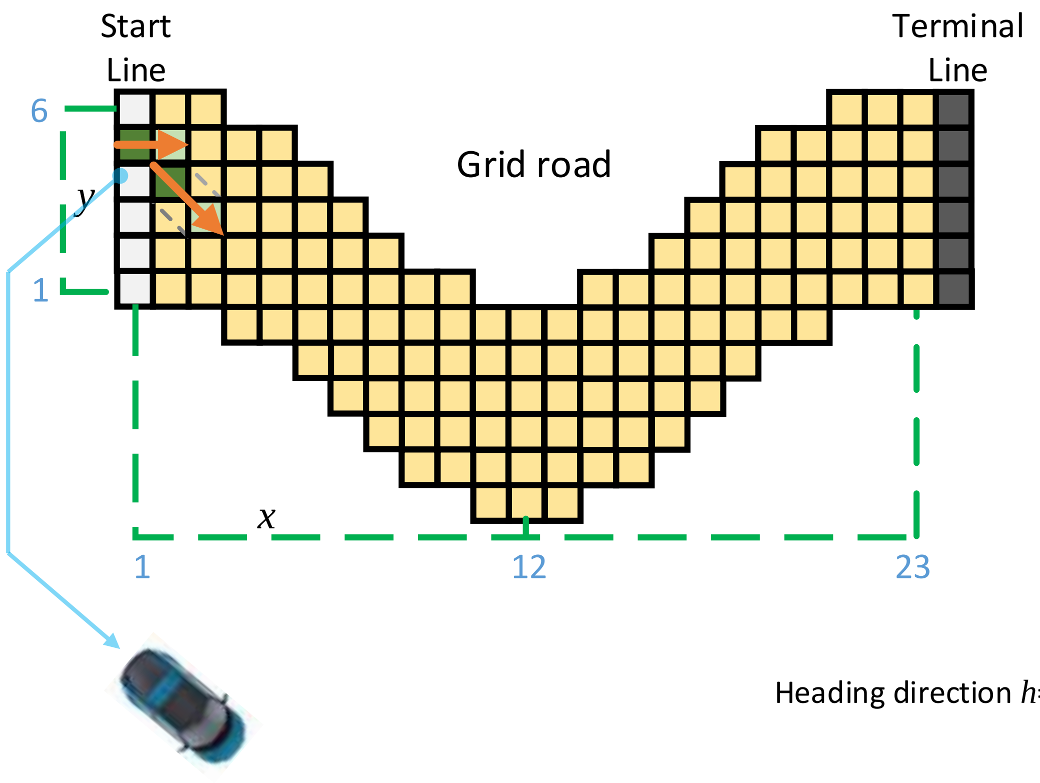
Example: Autonomous Driving

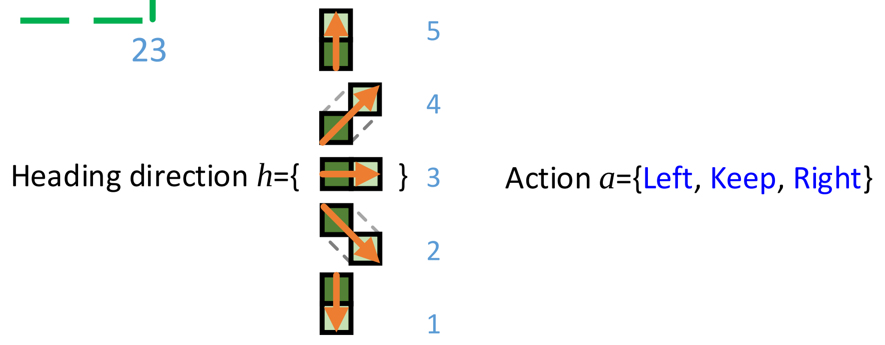




- Autonomous car works on a curved road
- Lane keeping control function with constant speed
- Autonomous car seeks to reach destination as soon as possible

Example: Autonomous Driving





- Each car is composed of two adjacent cells
- Every time instant, car will move one step forward along current heading direction

Example: Autonomous Driving

State space

$$s = [x, y, h]^{\mathrm{T}} \in \mathcal{S}$$
$$\mathcal{S} = \mathcal{S}_x \times \mathcal{S}_y \times \mathcal{S}_h$$
$$\mathcal{S}_x = \{x_{(1)}, x_{(2)}, \cdots, x_{(23)}\}, \mathcal{S}_y = \{y_{(1)}, y_{(2)}, \cdots, y_{(6)}\}, \mathcal{S}_h = \{h_{(1)}, h_{(2)}, \cdots, h_{(5)}\}$$

Action space:

$$\mathcal{A} = \{\mathrm{Left}, \mathrm{Keep}, \mathrm{Right}\}$$

- Each action can deterministically steer the car to a neighboring direction, and move the car on-step forward along current direction

$$\Pr\left\{s' = \left[x_{(2)}, y_{(4)}, h_{(2)}\right]^{\mathrm{T}} \middle| \ s = \left[x_{(1)}, y_{(5)}, h_{(3)}\right]^{\mathrm{T}}, a = \mathrm{Right}\right\} = 1$$

Example: Autonomous Driving

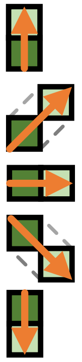
Reward

- Combine steering and moving rewards

$$r(s,a,s') = r_{\mathrm{Steer}} + r_{\mathrm{Move}}$$

$$r_{\mathrm{Steer}} = \begin{cases} -1 & \text{, if } a = \mathrm{Left} \\ 0 & \text{, if } a = \mathrm{Keep} \\ -1 & \text{, if } a = \mathrm{Right} \end{cases}$$

$$r_{\mathrm{Move}} = \begin{cases} -1 & \text{, if } h' = 1 \text{ or } 3 \text{ or } 5 \\ -\sqrt{2} & \text{, if } h' = 2 \text{ or } 4 \end{cases}$$

Heading direction h={  }

Action a={Left, Keep, Right}

<Reinforcement Learning and Control>