无人系统设计

课　程：软件工程专业-专业实践类课程
学　分：3
总课时：48

课程参考教材：

《认识飞行（第二版）》/《Understanding Flight，2nd》
作者： David F. Anderson, Scott Eberhardt
译者：周尧明（2019年） / 韩莲（2011年）
北京联合出版公司2019.07 / 航空工业出版社2011.01

授课教师：王赓
课程助教：李旭辉、蒋李康、方俊杰、张源娣、范文婷、曹恺洋、杨逍

课程主要内容

（1）认识飞行
- 牛顿力学（作用力与反作用力）
- 刚体转动（转矩、陀螺、进动） /大学物理基础

（2）认识多种多样的无人飞行系统
- 飞行原理
- 动力技术（螺旋桨、喷气式）

（3）控制技术
- 飞行操纵原理（机翼、襟翼、旋翼、尾桨、自动倾斜器）
- 作动器（电动机、舵机（PWM调制））
- 传感器（电子指南针、加速度计、陀螺仪、GPS、高度计、高速相机、全景相机、……）
- 电子控制器（PID算法、飞行控制原理与算法）

（4）飞行性能（飞行性能指标体系、稳定性、可靠性、易操作性）

课程主要内容

（5）基于4旋翼、固定翼模型机的认知验证实验
（含早期自由组合发现学习过程）

（6）仿真技术
- 飞行器建模（动力学、运动学）/大学物理基础、高等数学
- 软件技术（Unity3D、MATLAB/Simulink）

（7）仿真技术实践（半实物）
- 无人AI战机模拟格斗对抗系统
- 软件技术（Unity3D、MATLAB/Simulink、图像处理技术、人工智能AI技术、计算加速技术 ……）

（8）发挥想象力和所学的自由拓展设计（理论设计/尽量据情实验验证）

（9）课程综合设计与答辩

本讲内容：关于强化学习方法

王雨乐

wyl666@sjtu.edu.cn

张源娣

zydiii@sjtu.edu.cn

References and Acknowledgement

- Some slides in this lecture are from Prof. Weinan Zhang's course for machine Learning.

Weinan Zhang

Shanghai Jiao Tong University

http://wnzhang.net

Review-Machine Learning

- **Supervised Learning**
  - To perform the desired output given the data and labels
  - e.g., to build a loss function to minimize
  - Deep Learning, aka. Deep Supervised Learning
- **Unsupervised Learning**
  - To analyze and make use of the underlying data patterns/structures
  - e.g., to build a log-likelihood function to maximize
- **Reinforcement Learning**

Review-Supervised Learning

- Given the training dataset of (data, label) pairs,

$$D = \{(x_i, y_i)\}_{i=1,2,\ldots,N}$$

- Let the machine learn a function from data to label

$$y_i \simeq f_\theta(x_i)$$

- Learning is referred to as updating the parameter θ
- Learning objective: make the prediction close to the ground truth

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, f_\theta(x_i))$$

Review-Unsupervised Learning

- Given the training dataset

$$D = \{x_i\}_{i=1,2,\ldots,N}$$

  - Let the machine learn the data underlying patterns
- Sometimes build latent variables

$$z \longrightarrow x$$

Review-Unsupervised Learning

- Estimate the probabilistic density function (p.d.f.)

$$p(x;\theta) = \sum_z p(x|z;\theta)p(z;\theta)$$

- Learning is also referred to as updating the parameter θ
- Maximize the log-likelihood of training data

$$\max_\theta \frac{1}{N}\sum_{i=1}^{N} \log p(x;\theta)$$

  - e.g. BERT

Two Kinds of Machine Learning

Prediction

- Predict the desired output given the data (supervised learning)
- Generate data instances (unsupervised learning)

Decision Making

- Take actions based on a particular state in a dynamic environment (reinforcement learning)
  - to transit to new states
  - to receive immediate reward
  - to maximize the accumulative reward over time
- Learning from interaction

Machine Learning Categories

- Supervised Learning
  - To perform the desired output given the data and labels

$$p(y|x)$$

- Unsupervised Learning
  - To analyze and make use of the underlying data patterns/structures

$$p(x)$$

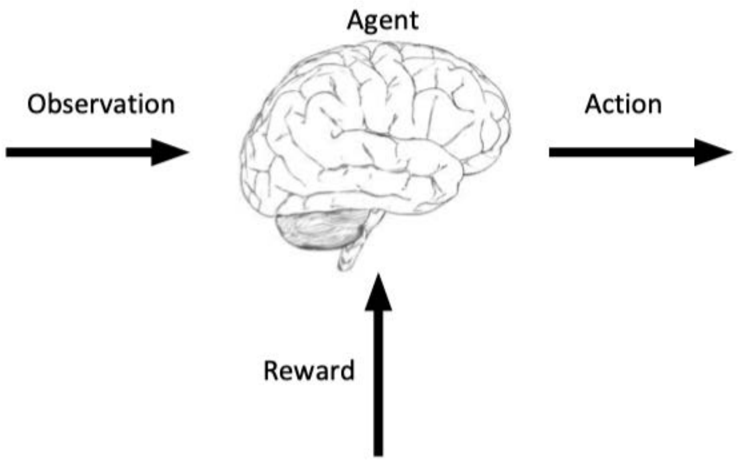
- Reinforcement Learning
  - To learn a policy of taking actions in a dynamic environment and acquire rewards

$$\pi(a|x)$$

Contents

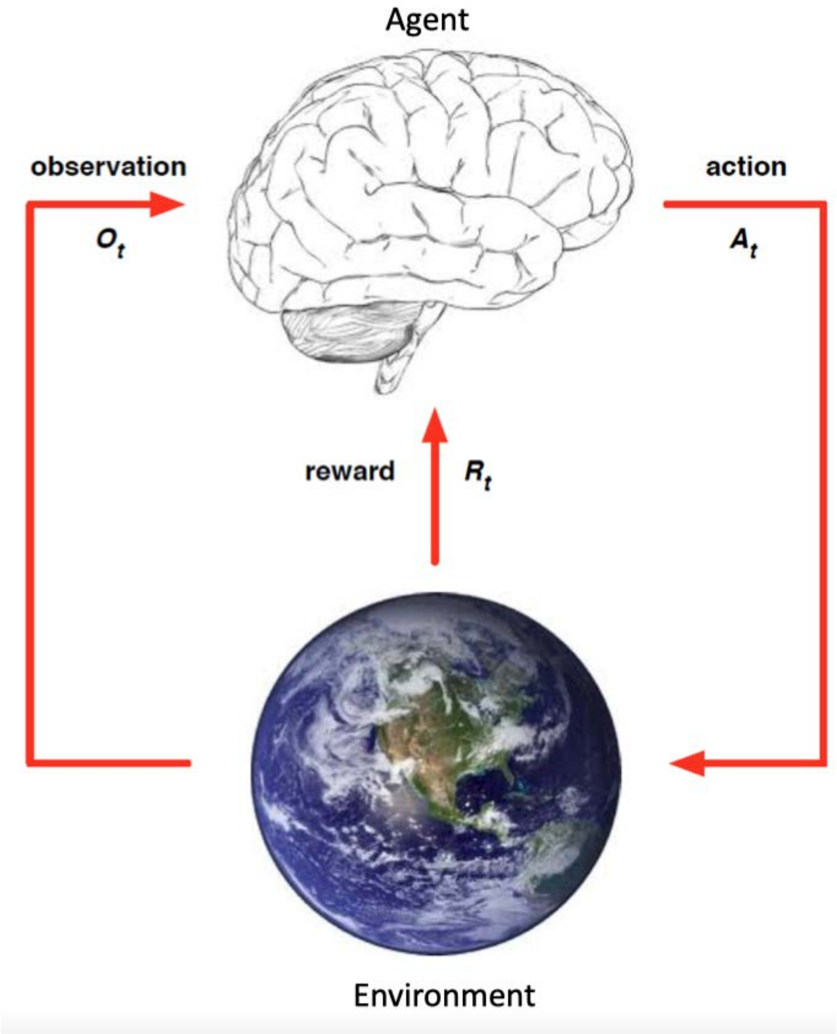
Reinforcement Learning Definition

- A computational approach by learning from interaction to achieve a goal



- Three aspects
  - Sensation: sense the state of the environment to some extent
  - Action: able to take actions that affect the state and achieve the goal
  - Goal: maximize the cumulative reward over time

Reinforcement Learning



- At each step t, the agent
  - Receives observation O_t
  - Receives scalar reward R_t
  - Executes action A_t

- The environment
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}

Elements of RL Systems

- History is the sequence of observations, action, rewards

$$H_t = O_1, R_1, A_1, O_2, R_2, A_2, \ldots, O_{t-1}, R_{t-1}, A_{t-1}, O_t, R_t$$

  - i.e. all observable variables up to time t
  - E.g., the sensorimotor stream of a robot or embodied agent
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- State is the information used to determine what happens next (actions, observations, rewards) Formally, state is a function of the history:

$$S_t = f(H_t)$$

Elements of RL Systems

- Policy is the learning agent's way of behaving at a given time
  - It is a map from state to action
  - Deterministic policy

$$a = \pi(s)$$

  - Stochastic policy

$$\pi(a|s) = P(A_t = a | S_t = s)$$

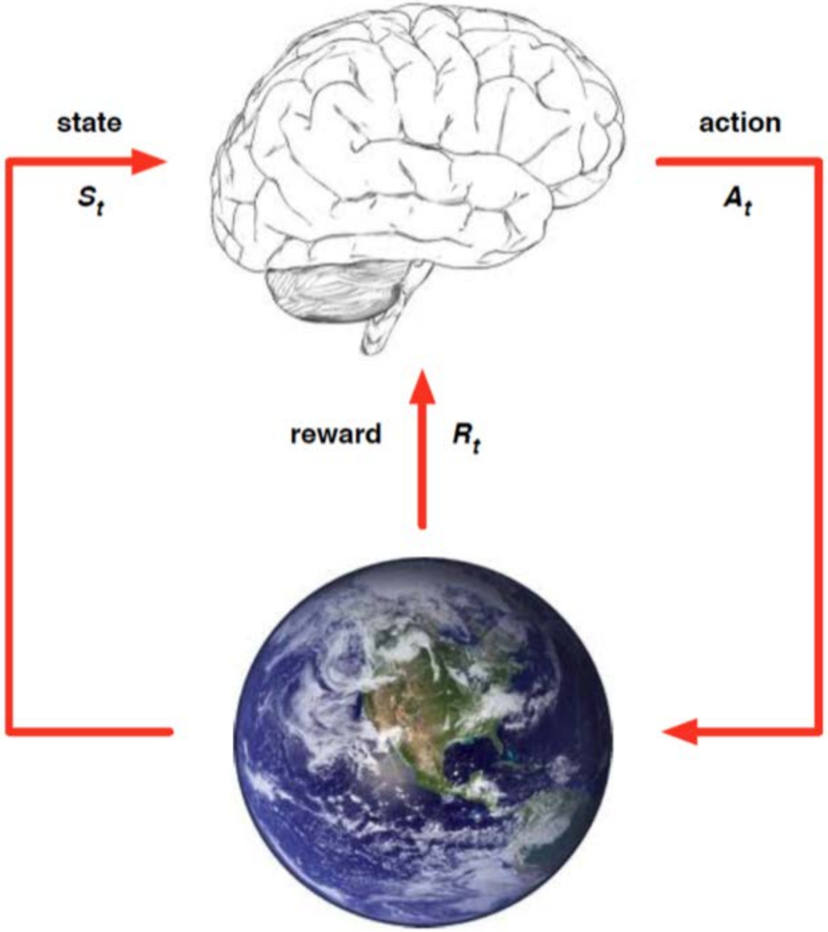
Elements of RL Systems

- A **Model** of the environment that mimics the behavior of the environment
  - Predict the next state

$$\mathcal{P}_{sa}(s') = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

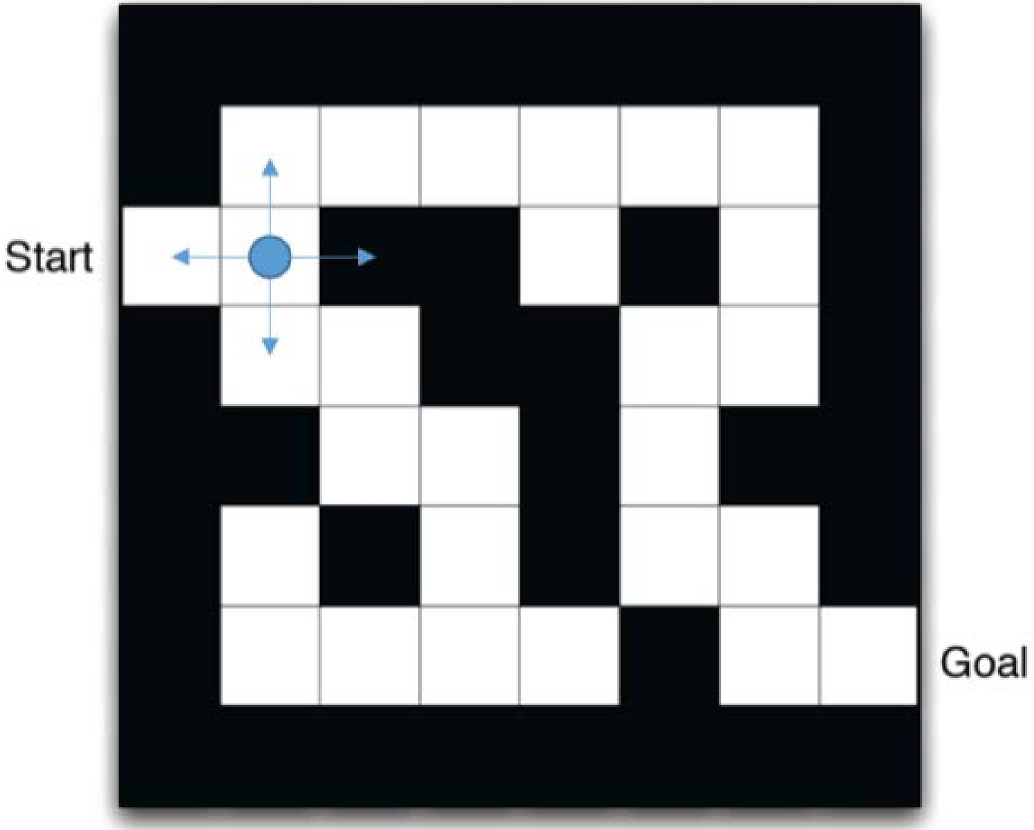
- Predicts the next (immediate) reward

$$\mathcal{R}_s(a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$



Elements of RL Systems

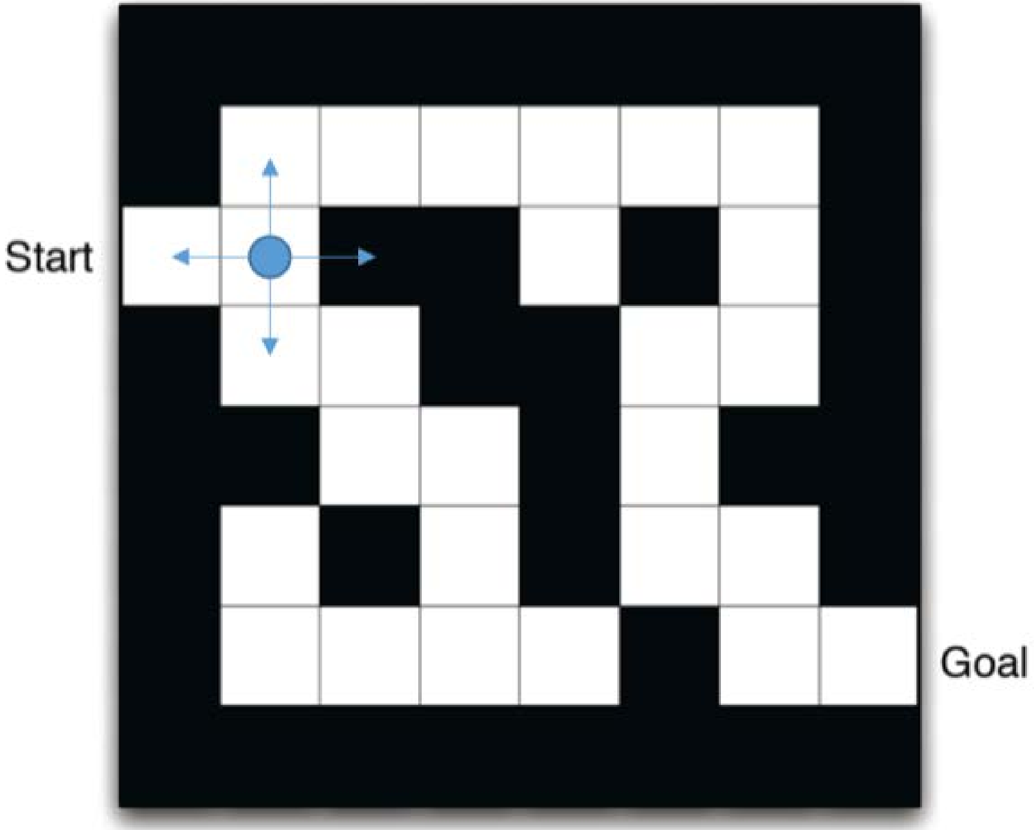
- Maze Example



- State: agent’s location
- Action: N,E,S,W

Elements of RL Systems

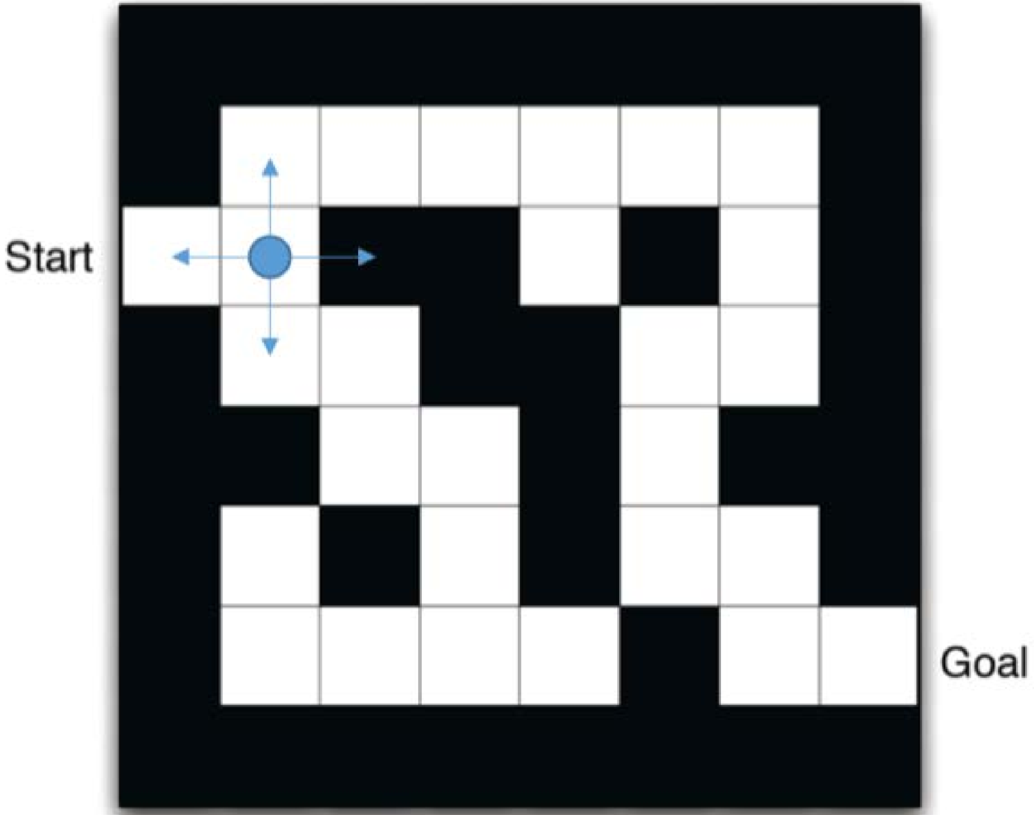
- Maze Example



- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
  - No move if the action is to the wall

Elements of RL Systems

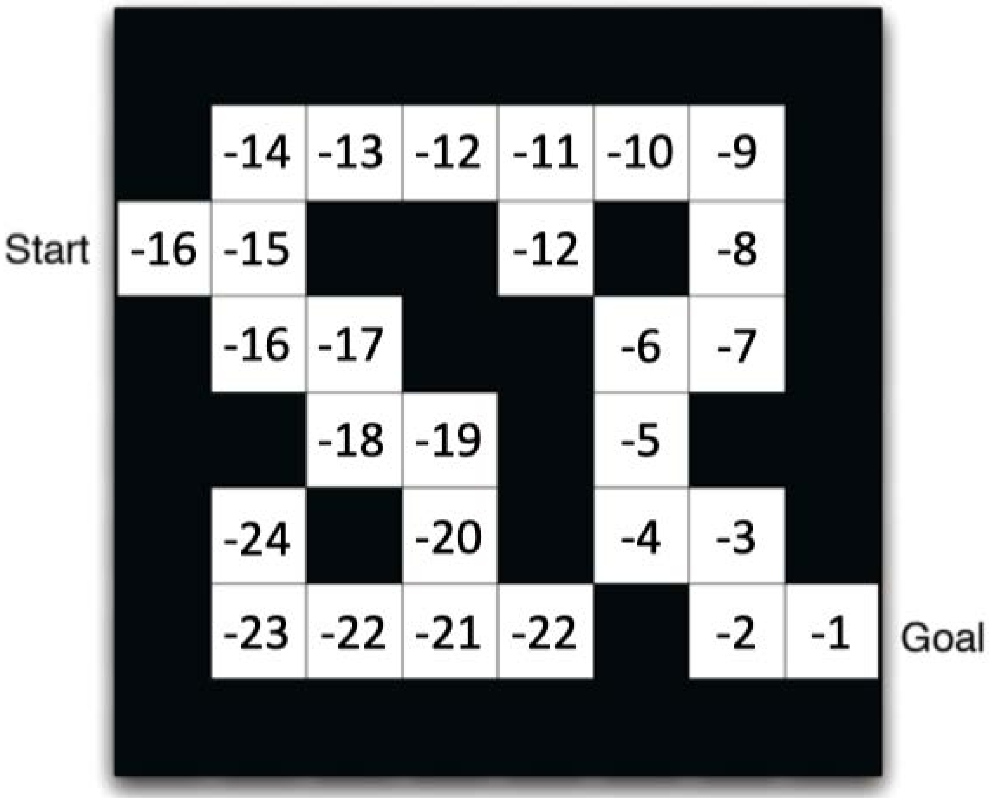
- Maze Example



- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
  - No move if the action is to the wall
- Reward: -1 per time step

Elements of RL Systems

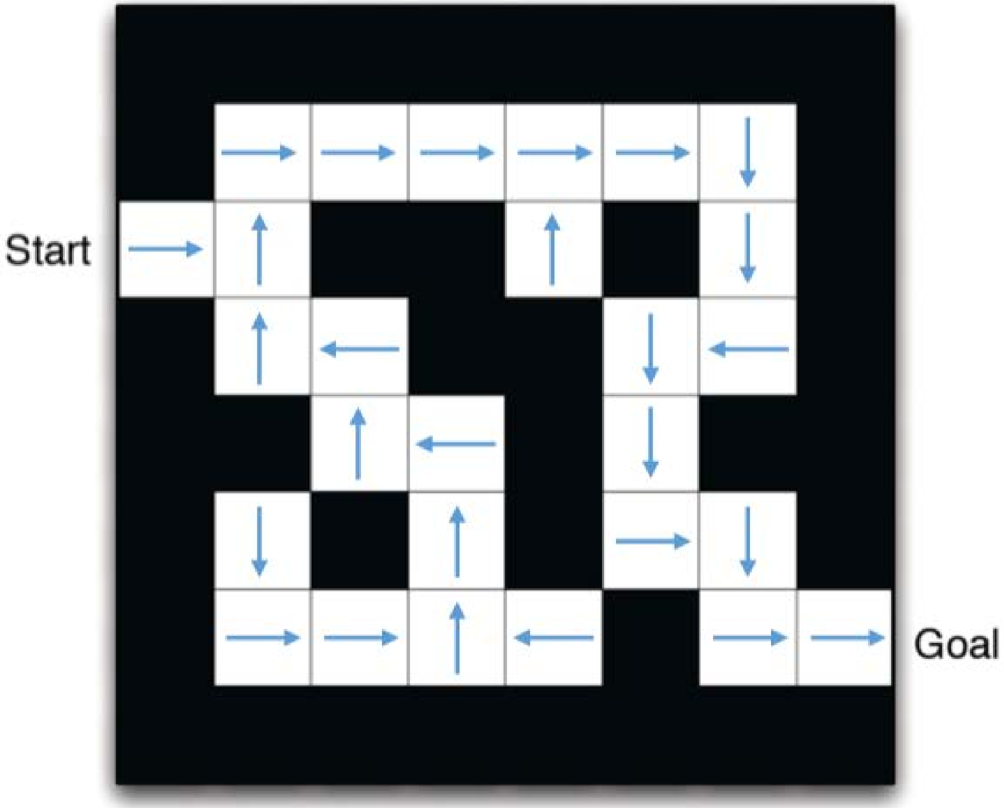
- Maze Example



- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
  - No move if the action is to the wall
- Reward: -1 per time step
- Numbers represent value $v_\pi(s)$ of each state s.

Elements of RL Systems

- Maze Example



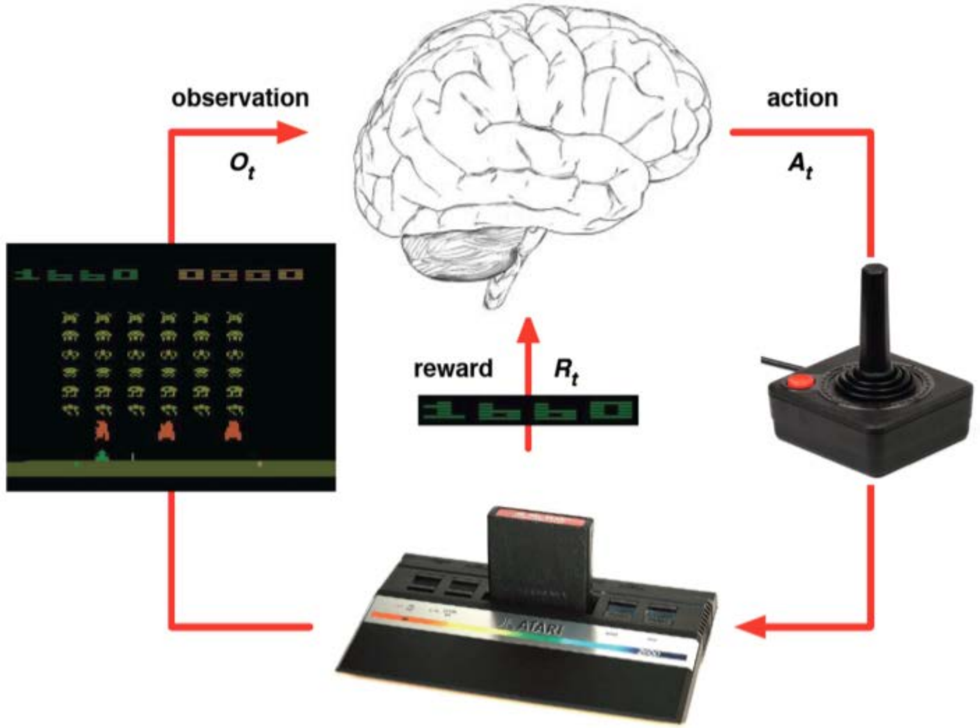
- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
  - No move if the action is to the wall
- Reward: -1 per time step
- The given policy
  - Arrows represent policy $\pi(s)$ for each state s.

Categorizing RL Agents

- Model based RL
  - Policy and/or value function
  - Model of the environment
  - E.g., the maze game, game of Go
- Model-free RL
  - Policy and/or value function
  - No model of the environment
  - E.g., general playing Atari games

Atari Example

- Rules of the game are unknown
- Learn from interactive game-play
- Pick actions on joystick, see pixels and scores



Categorizing RL Agents

- Model based RL
  - Markov Decision Process
  - Planning by Dynamic Programming
- Model-free RL
  - Policy and/or value function
  - No model of the environment
  - E.g., general playing Atari games

Markov Decision Process

- Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.

- MDPs formally describe an environment for RL
  - where the environment is FULLY observable
  - i.e. the current state completely characterizes the process (Markov property)

Markov Property

- "The future is independent of the past given the present"
- Definition
  - A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

- Properties
  - The state captures all relevant information from the history
  - Once the state is known, the history may be thrown away
  - i.e. the state is sufficient statistic of the future

Markov Decision Process

- A Markov decision process is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$
- S is the set of states
  - E.g., location in a maze, or current screen in an Atari game
- A is the set of actions
  - E.g., move N, E, S, W, or the direction of the joystick and the buttons
- P_{sa} are the state transition probabilities
  - For each state $s \in S$ and action $a \in A$, P_{sa} a distribution over the next state in S
- $\gamma \in [0,1]$ is the discount factor for the future reward
- R is the reward function

Markov Decision Process

- The dynamics of an MDP proceeds as
  - Start in a state S_0
  - The agent chooses some action $a_0 \in A$
  - The agent gets the reward $R(s_0, a_0)$
  - MDP randomly transits to some successor state $s_1 \sim P_{s0a0}$
- This proceeds iteratively

$$s_0 \xrightarrow[R(s_0,a_0)]{a_0} s_1 \xrightarrow[R(s_1,a_1)]{a_1} s_2 \xrightarrow[R(s_2,a_2)]{a_2} s_3 \cdots$$

- The total payoff of the agent is

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$$

Markov Decision Process

- For a large part of cases, reward is only assigned to the state
  - E.g., in maze game, the reward is on the location
  - In game of Go, the reward is only based on the final territory
- The reward function R(s)
- MDPs proceed

$$s_0 \xrightarrow[R(s_0)]{a_0} s_1 \xrightarrow[R(s_1)]{a_1} s_2 \xrightarrow[R(s_2)]{a_2} s_3 \cdots$$

- cumulative reward (total payoff)

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

Reward on State Only

- The goal is to choose actions over time to maximize the expected cumulative reward

$$\mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$

- $\gamma \in [0,1]$ is the discount factor for the future reward, which makes the agent prefer immediate reward to future reward
  - In finance case, today's $1 is more valuable than $1 in tomorrow
- Given a particular policy $\pi(s)$
- Define the value function for π:

$$V^\pi(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$

  - i.e. expected cumulative reward given the start state and taking actions according to π

Bellman Equation for Value Function

- Define the value function for π

$$V^\pi(s) = \mathbb{E}[R(s_0) + \underbrace{\gamma R(s_1) + \gamma^2 R(s_2) + \cdots}_{\gamma V^\pi(s_1)} | s_0 = s, \pi]$$

$$= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s')$$

Bellman Equation

Immediate Reward

Time decay

State transition

Value of the next state

Optimal Value Function

- The optimal value function for each state s is best possible sum of discounted rewards that can be attained by any policy

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- The Bellman's equation for optimal value function and the optimal policy

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- For every state s and every policy π

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

Value Iteration & Policy Iteration

- Note that the value function and policy are correlated

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

- It is feasible to perform iterative update towards the optimal value function and optimal policy
  - Value iteration
  - Policy iteration

Value Iteration

- For an MDP with finite state and action spaces

$$|S| < \infty, |A| < \infty$$

- Value iteration is performed as

1. For each state s, initialize V(s) = 0.
2. Repeat until convergence {

   For each state, update

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s')V(s')$$

   }

Synchronous vs. Asynchronous

- Synchronous value iteration stores two copies of value functions
  - For all s in S

$$V_{\text{new}}(s) \leftarrow \max_{a \in A}\left(R(s) + \gamma \sum_{s' \in S} P_{sa}(s') V_{\text{old}}(s')\right)$$

  - Update $V_{\text{old}}(s') \leftarrow V_{\text{new}}(s)$
- In-place asynchronous value iteration stores one copy of value function
  - For all s in S

$$V(s) \leftarrow \max_{a \in A}\left(R(s) + \gamma \sum_{s' \in S} P_{sa}(s') V(s')\right)$$

Value Iteration Example: Shortest Path

- Setting: Reward is -1 for every step

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

Policy Iteration

- For an MDP with finite state and action spaces

$$|S| < \infty, |A| < \infty$$

- Policy iteration is performed as
  - Initialize π randomly
  - Repeat until convergence {
    - Let $V := V^{\pi}$
    - For each state, update

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

Policy Iteration

- For an MDP with finite state and action spaces

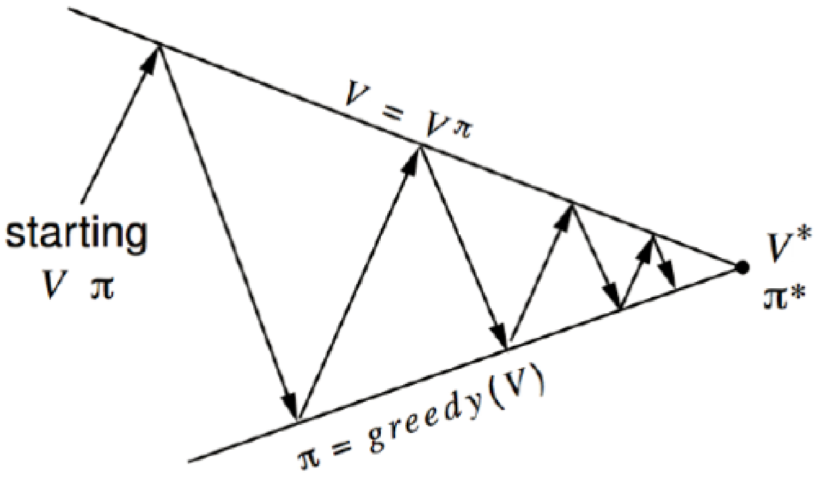
$$|S| < \infty, |A| < \infty$$

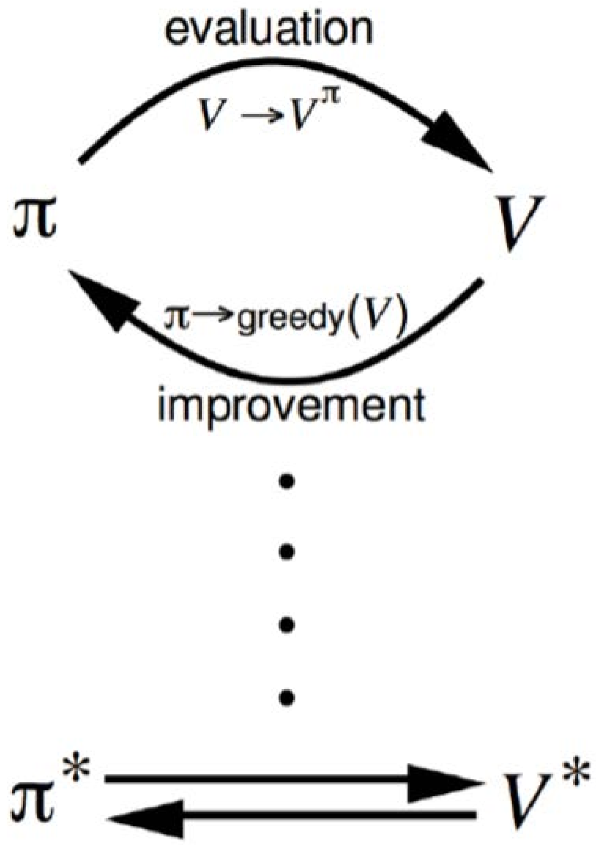
- Policy iteration is performed as
  - Initialize π randomly
  - Repeat until convergence {
    - Let $\boxed{V := V^{\pi}}$
    - For each state, update

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')V(s')$$

- The step of value function update could be time-consuming

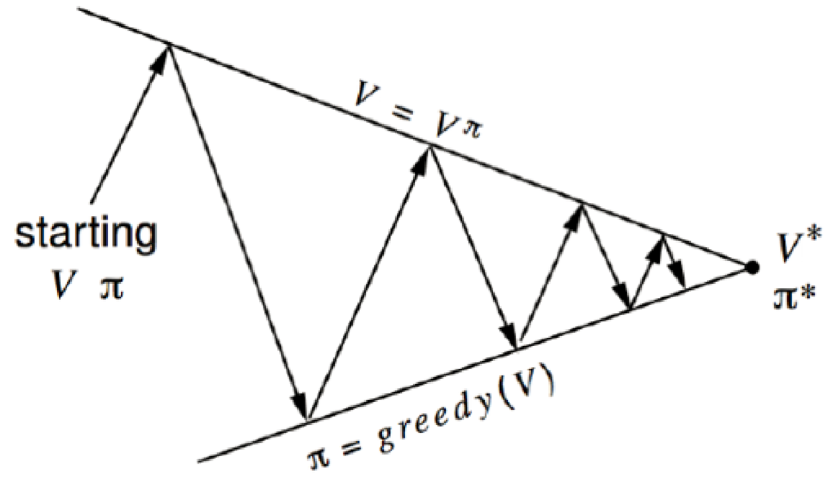
Policy Iteration

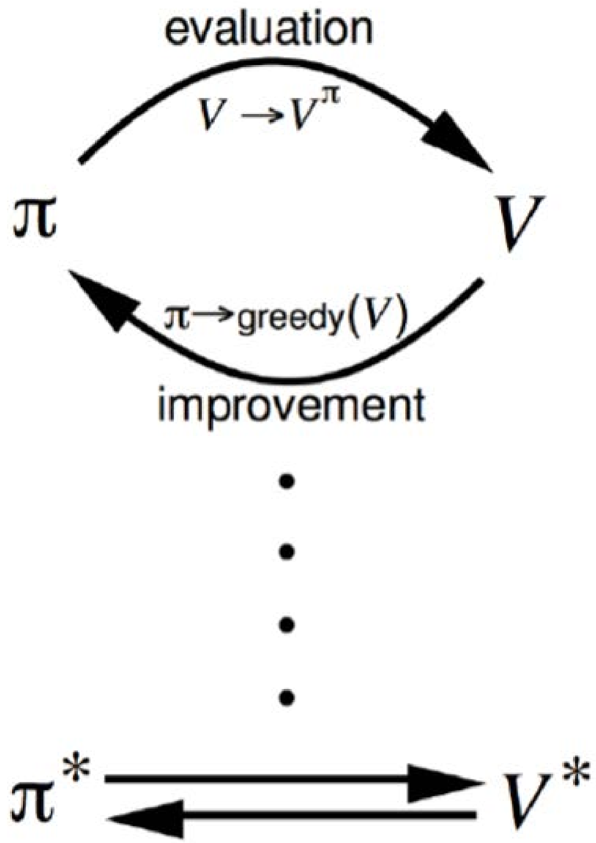




- Policy Iteration
  - Estimate V^π
  - Iterative policy evaluation

Policy Iteration





- Policy Improvement
  - Generate $\pi' \geq \pi$
  - Greedy Policy Improvement

Value Iteration vs. Policy Iteration

Value iteration

1. For each state *s*, initialize *V*(*s*) = 0.
2. Repeat until convergence {

   For each state, update

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s')$$

}

Policy iteration

1. Initialize π randomly
2. Repeat until convergence {
   a) Let $V := V^{\pi}$
   b) For each state, update

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

}

Value Iteration vs. Policy Iteration

- Remarks:
  - Value iteration is a greedy update strategy
  - In policy iteration, the value function update by bellman equation is costly
  - For small-space MDPs, policy iteration is often very fast and converges quickly
  - For large-space MDPs, value iteration is more practical (efficient)
  - Value iteration is like SGD and policy iteration is like BGD

Reinforcement Learning Materials

- Our slides on RL is mainly based on the materials from these masters.

Prof. Richard Sutton

- University of Alberta, Canada
- http://incompleteideas.net/sutton/index.html
- Reinforcement Learning: An Introduction (2nd edition)
- http://www.incompleteideas.net/book/the-book-2nd.html

Dr. David Silver

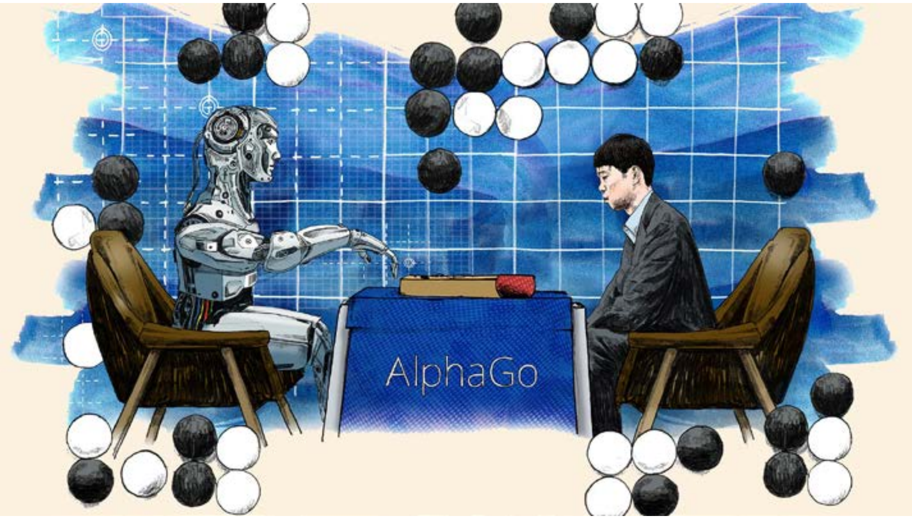
- Google DeepMind and UCL, UK
- http://www0.cs.ucl.ac.uk/staff/d.silver/web/Home.html
- UCL Reinforcement Learning Course
- http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Prof. Andrew Ng

- Stanford University, US
- http://www.andrewng.org/
- Machine Learning (CS229) Lecture Notes 12: RL
- http://cs229.stanford.edu/materials.html

Matlab 中的强化学习

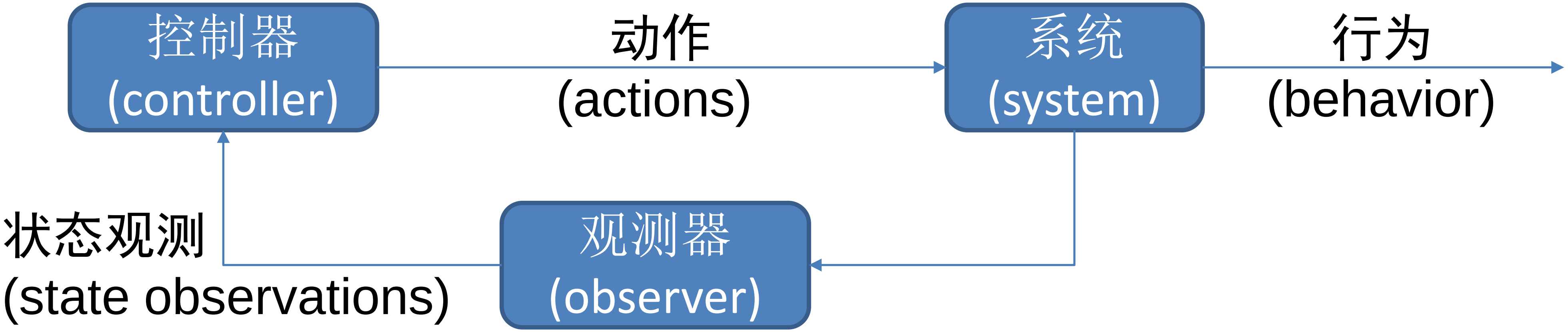
- 回顾强化学习基础知识，了解强化学习的基本工作流程
- 学习如何在 Matlab 中进行强化学习
- demo 讲解




什么是强化学习？

强化学习旨在学习如何做，即如何根据情况采取动作，从而实现数值奖励信号最大化。学习者不会接到动作指令，而是必须自行尝试去发现回报最高的动作方案。

—— Sutton and Barto，*Reinforcement Learning: An Introduction*

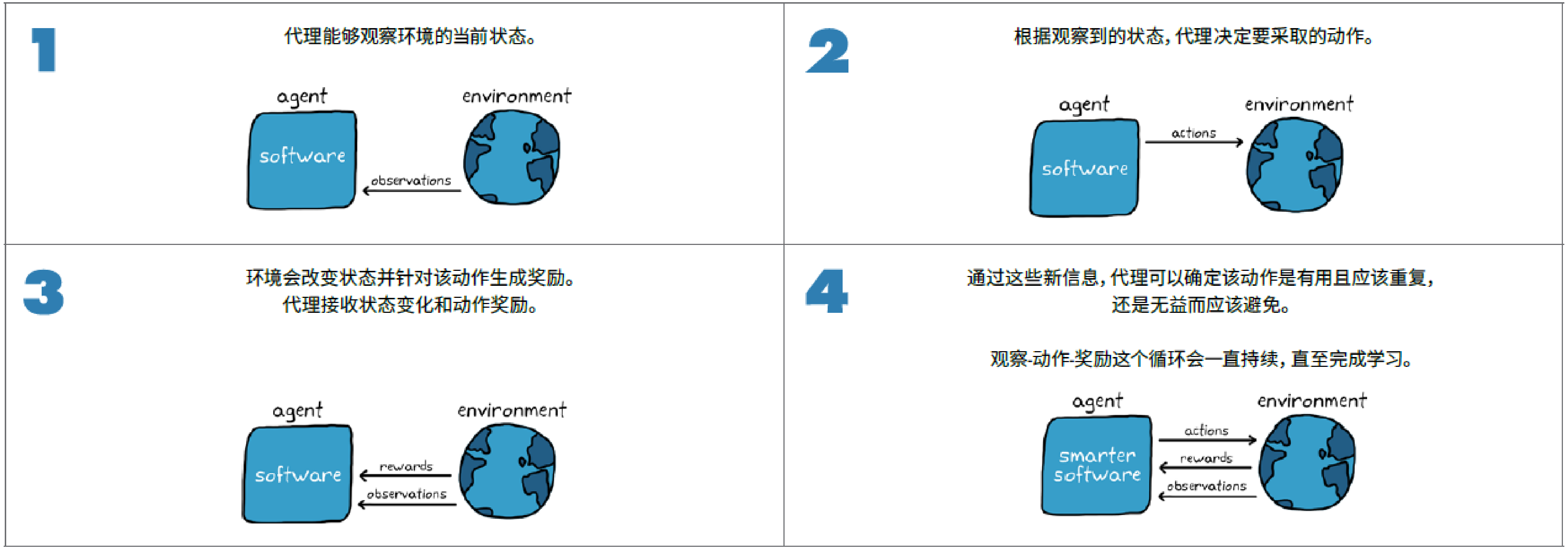


强化学习的优点

- 如果一个问题可以描述成或转化成序列决策问题，可以对状态、动作、奖赏进行定义，那么强化学习很可能可以帮助解决这个问题。
- 强化学习有可能帮助自动化、最优化手动设计的策略。
- 强化学习考虑序列问题，具有长远眼光，考虑长期回报，这种长远眼光对很多问题找到最优解非常关键。

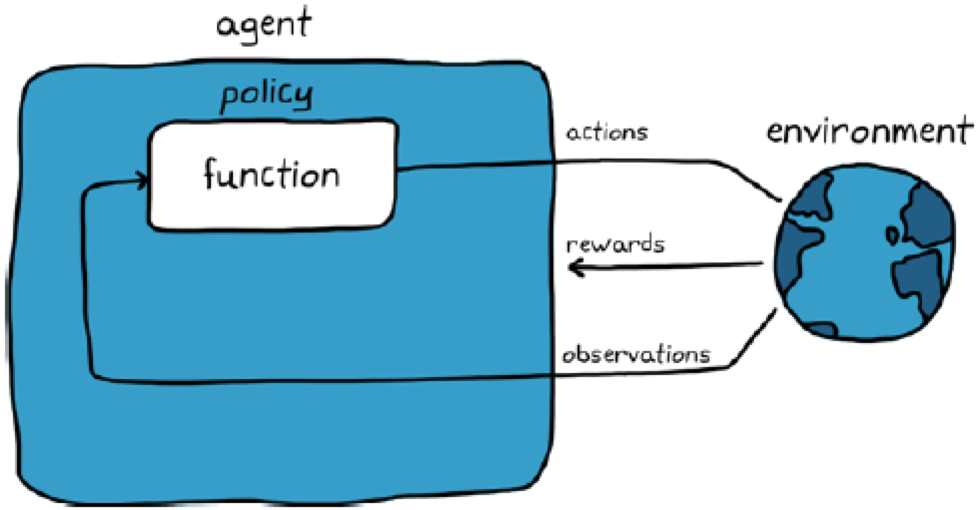
强化学习概述

- 强化学习采用动态环境数据，其目标是确定生成最优结果的最佳动作序列。
- 强化学习通过一个软件（代理）来探索环境、与环境交互并从环境中学习。



强化学习概述

- 代理中有一个函数可接收状态观测量（输入），并将其映射到动作集（输出）。此函数称为策略（policy），策略根据一组给定的观测量决定要采取的动作。
- 环境将生成奖励（rewards），向代理反映采取该动作指令的效果。
- 代理使用强化学习算法学习最佳环境交互策略，根据已采取的动作、环境状态观测量以及获得的奖励值来改变策略。



强化学习工作流程概述

1 您需要一个环境，供您的代理开展学习。您需要选择环境里应该有什么，是仿真还是物理设置。

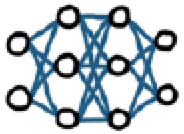
environment

2 您需要考虑最终想要代理做什么工作，并设计奖励函数，激励代理实现目标。

reward

3 您需要选择一种表示策略的方法。思考您想如何构造参数和逻辑，由此构成代理的决策部分。

policy

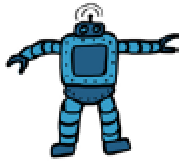


4 您需要选择一种算法来训练代理，争取找到最优的策略参数。

training

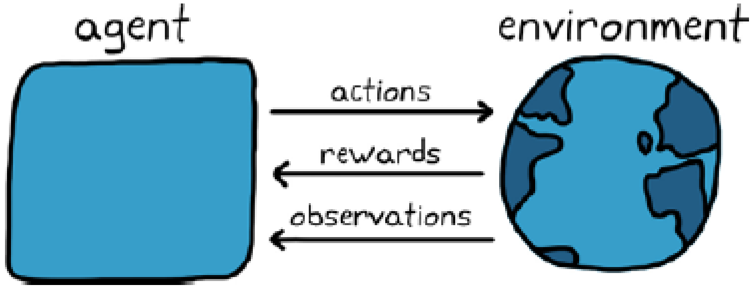
5 最后，您需要在实地部署该策略并验证结果，从而利用该策略。

deploy

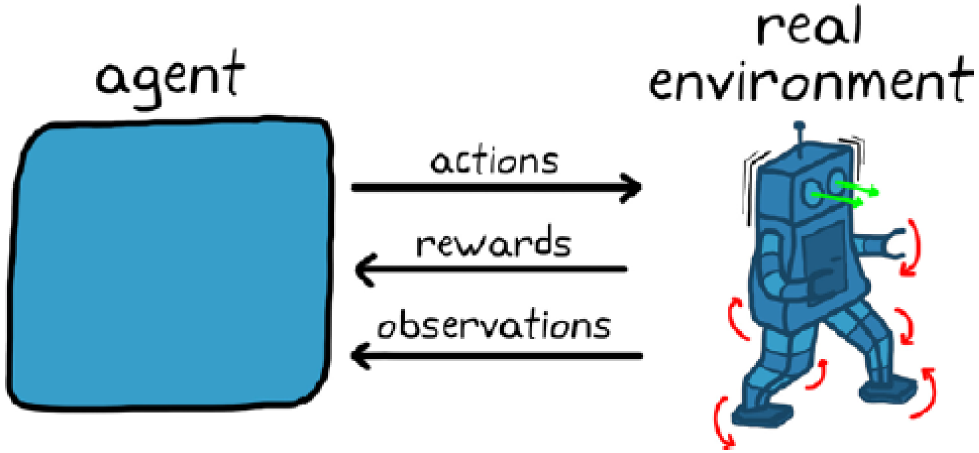


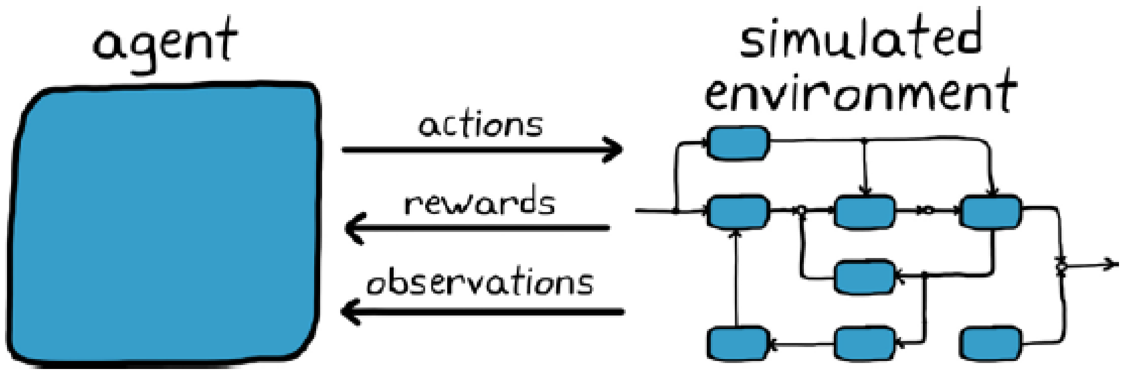
环境

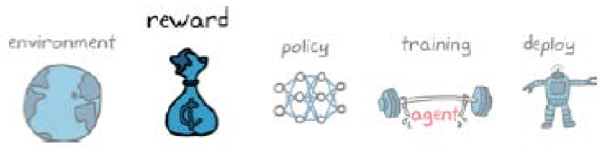
■ 环境是指存在于代理之外的一切元素。它既是代理动作产生作用的地方，又能生成奖励和观测量。



■ 环境可以是真实环境或仿真环境。

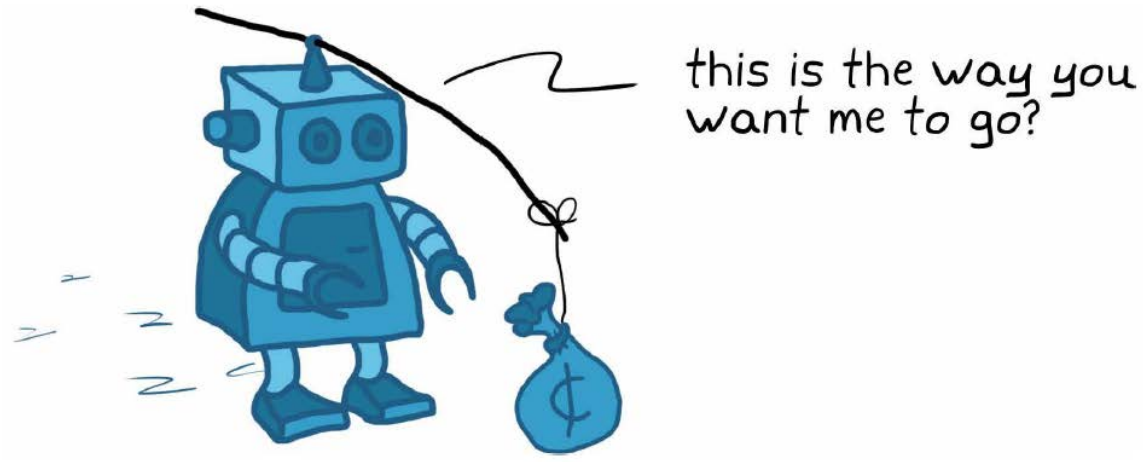






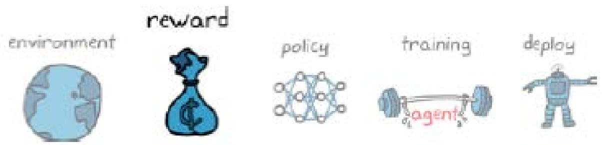
奖励

- 奖励让学习算法“明白”什么情况下策略变得更好，最终趋向目标结果。



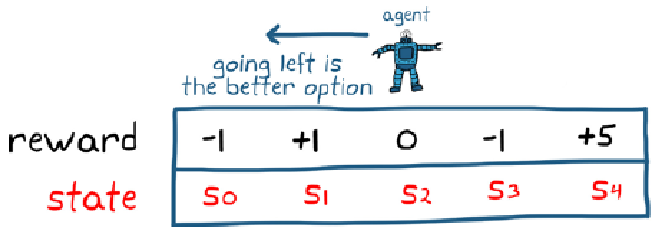
- 奖励是一个函数，会生成一个标量，代表处于某个特定状态并采取特定动作的代理的“优度”。在强化学习中对于创建奖励函数没有任何限制。可以采用稀疏奖励，或在每个时间步长后奖励，或者仅在较长一段时间后一个片段完全结束时给予奖励。奖励可以使用非线性函数计算，也可以通过几千个参数来计算。这完全取决于采取什么方式才能有效地训练代理。

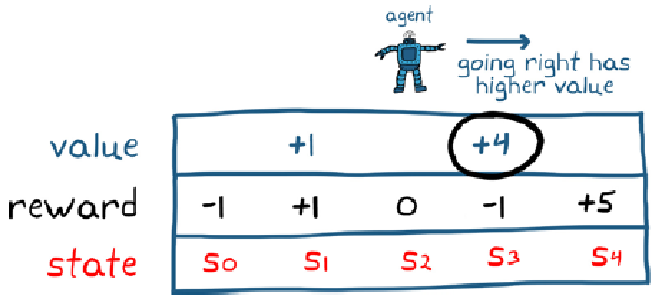
reward = function (state, action)



价值

- 评估一个状态或动作的价值，而不是奖励，可以帮助代理选择将会在一段时间内收取最多奖励（而不是短期利益）的动作。奖励是处于某一状态或采取特定动作的即时收益，价值是代理预期从某一状态和往后将会获得的总回报。



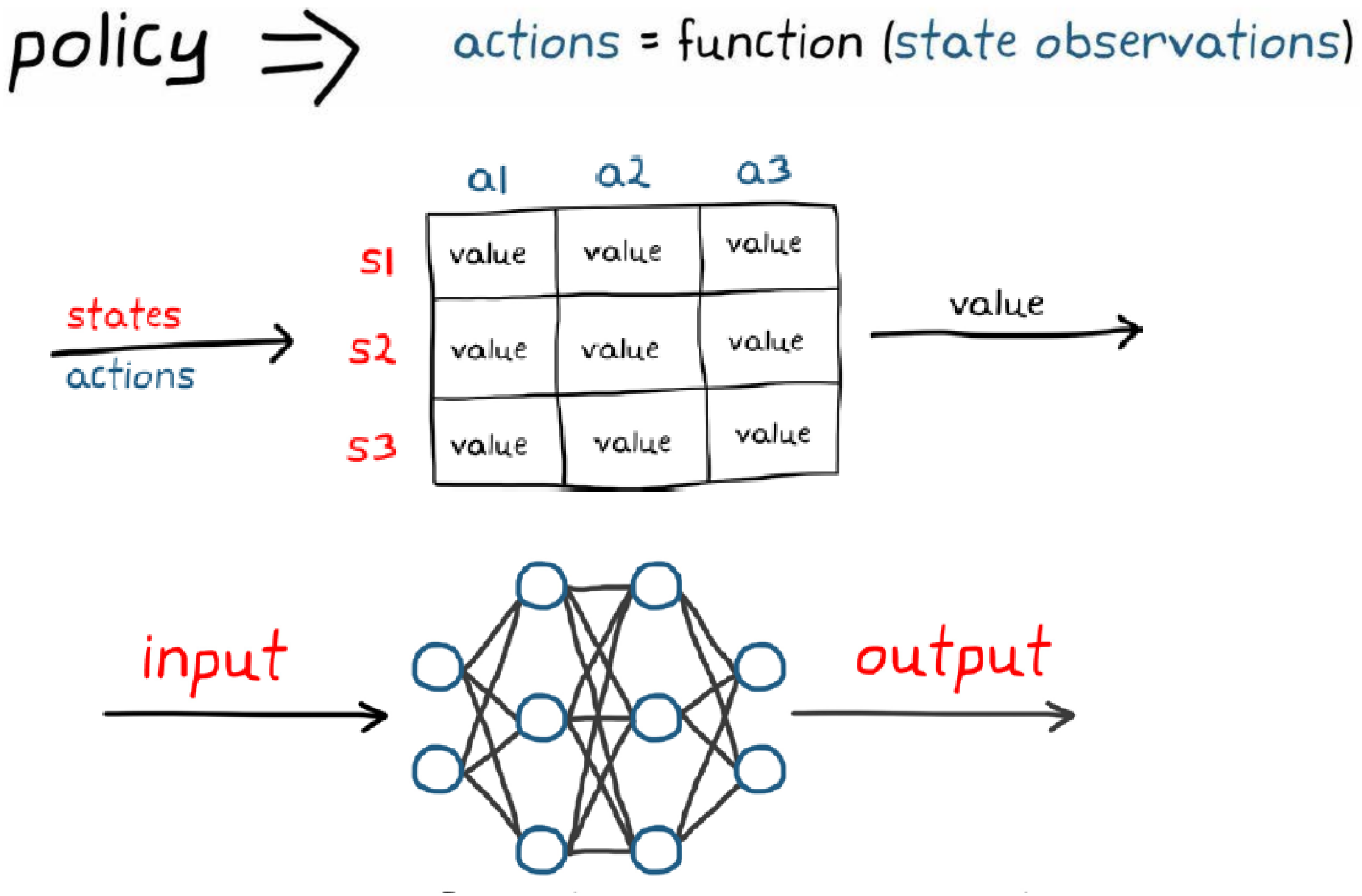


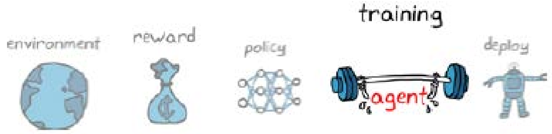
- 对更远的将来所能获取的奖励的预测不大可靠，在强化学习中，通过对奖励打折，越远的未来折扣越大，可以设置折扣系数 gamma，介于 0 到 1 之间。

$$\text{total discounted reward} = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots = \sum_{i=1}^{T} \gamma^{i-1} r_i$$

策略

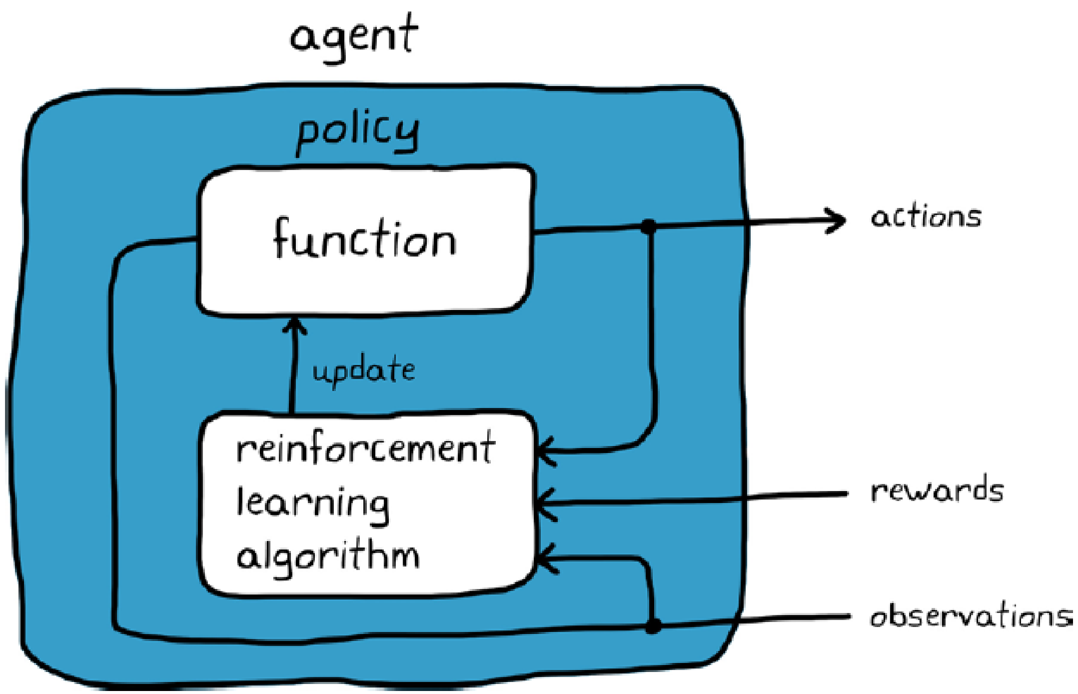
■ 策略是将观测量映射到动作的函数，需要通过学习算法进行优化。

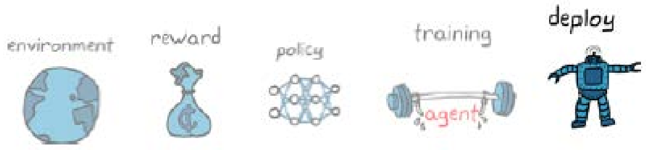




训练

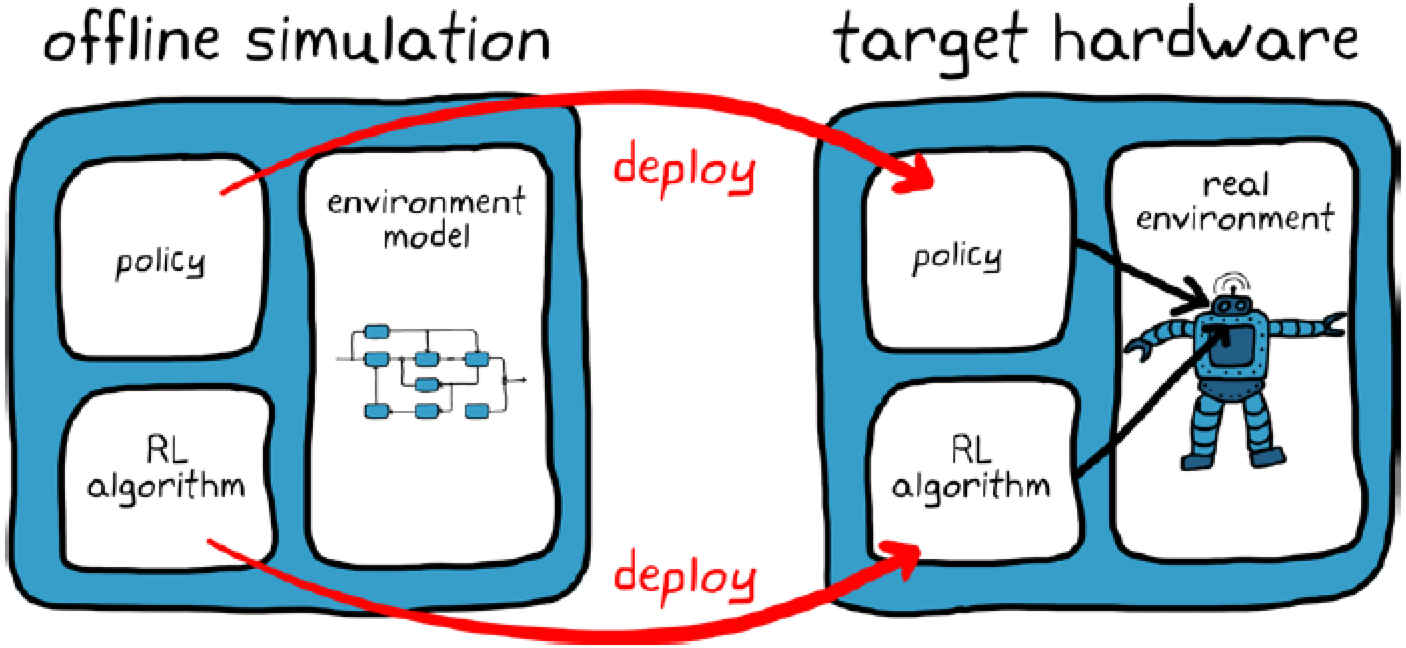
- 学习算法通过代理与环境不断交互的结果构建最优策略。





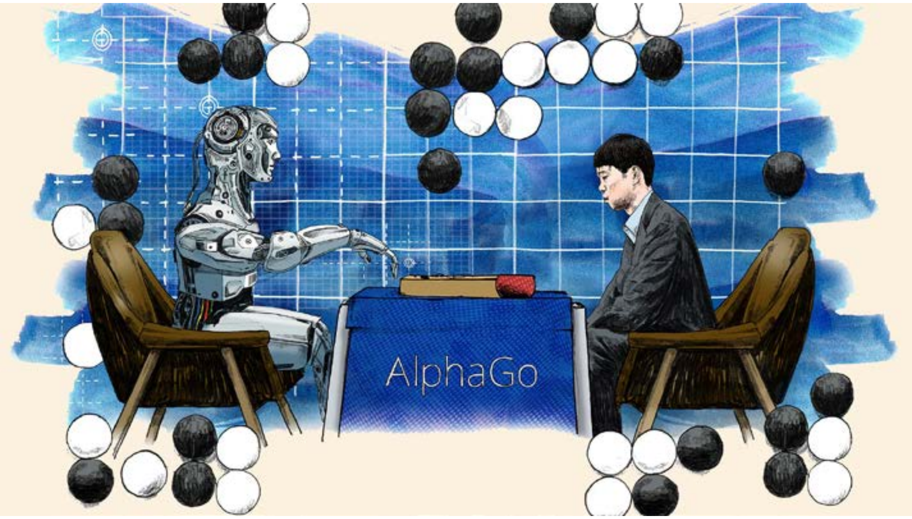
部署

- 如果在仿真环境中进行学习，最后需要将训练好的策略部署到目标硬件上，如果部署后仍然可能需要使用真实物理硬件继续开展学习，还需要将学习算法也部署到目标硬件上。



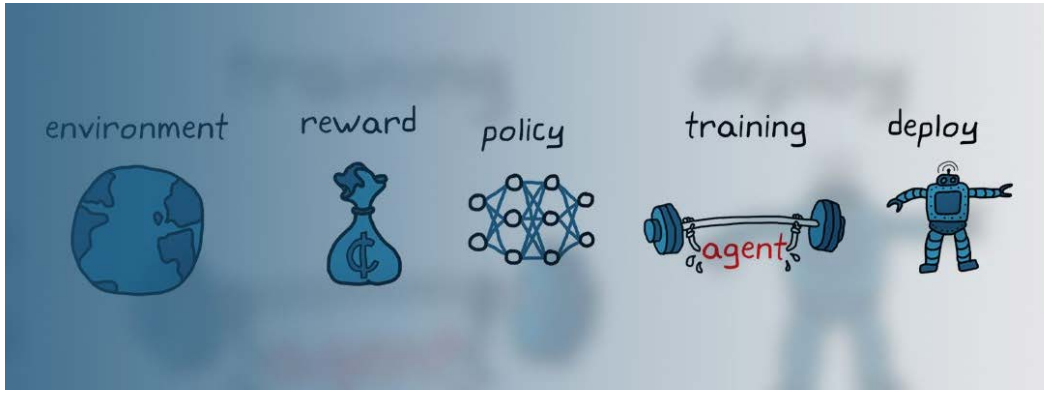
Matlab 中的强化学习

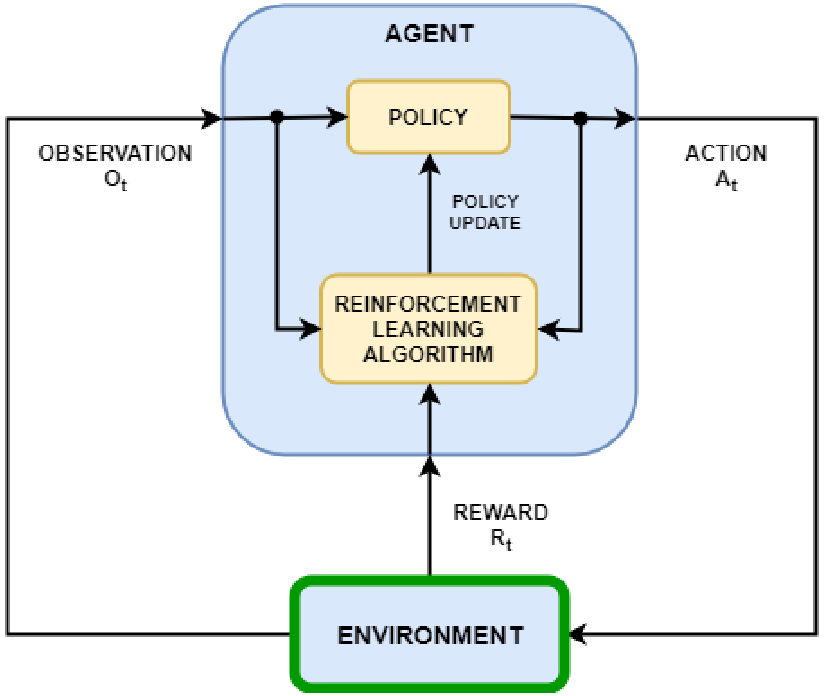
- 回顾强化学习基础知识，了解强化学习的基本工作流程
- 学习如何在 Matlab 中进行强化学习
- demo 讲解



主要流程

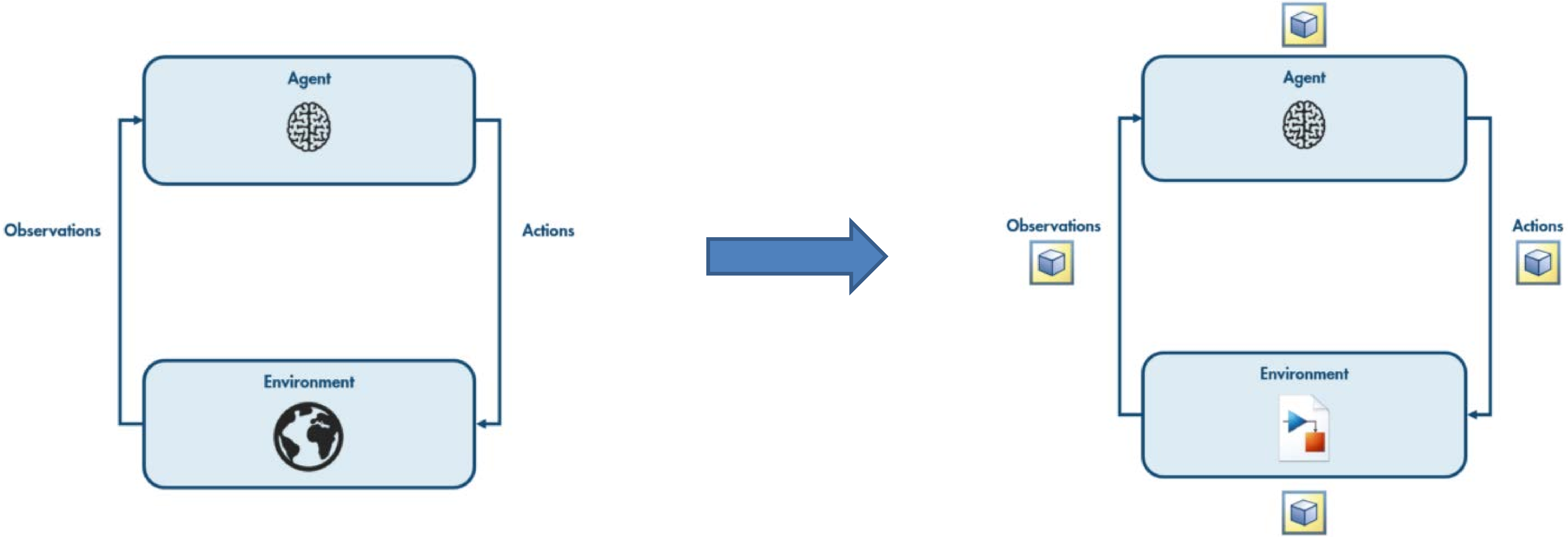
- 定义环境，包括需要模拟的环境，Observation 和 Action 接口，Reward 计算函数等
- 定义 Agent，包括需要采取的策略结构，学习算法等
- 使用定义好的环境和 Agent 进行训练





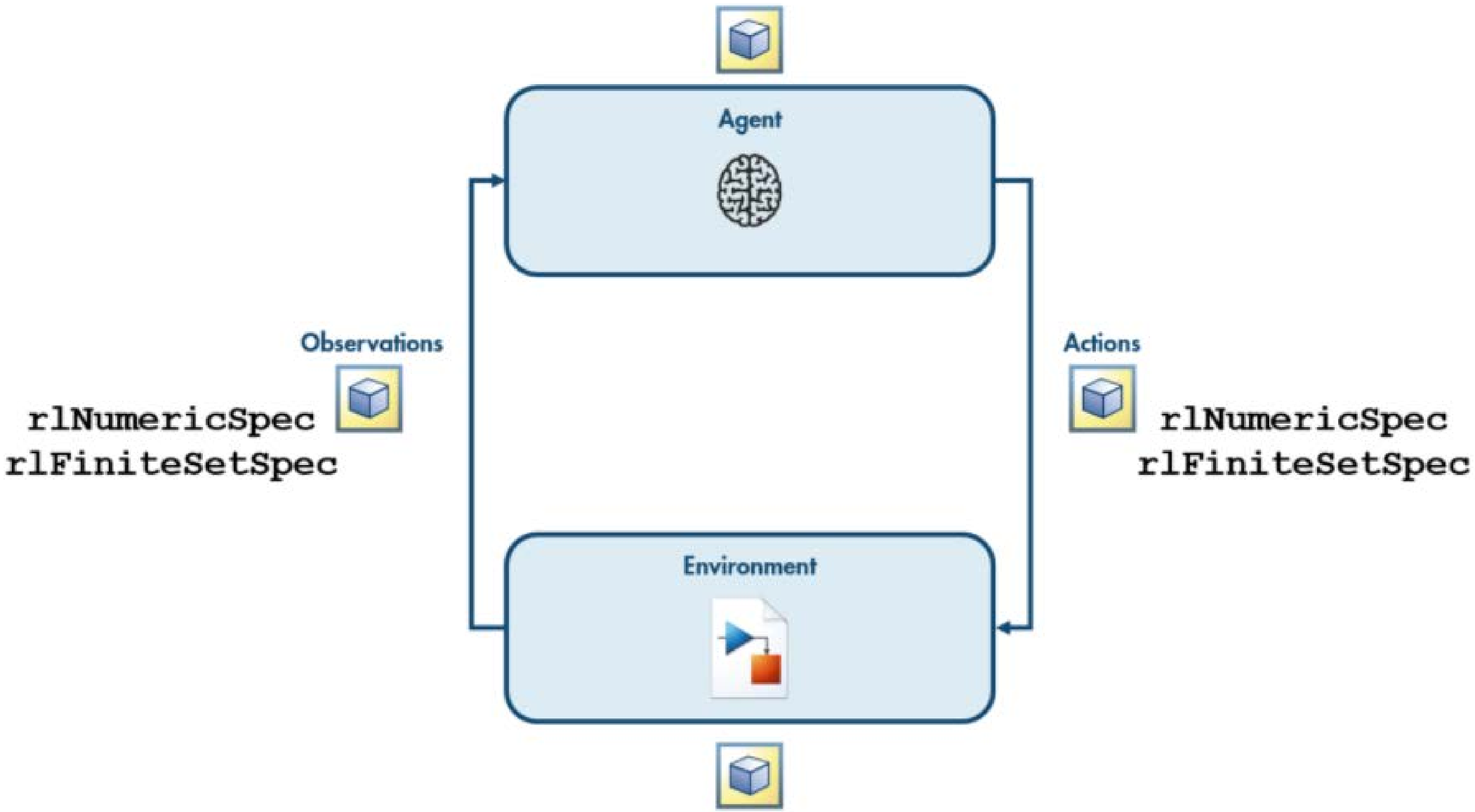
定义环境

- Agent 通过观测值做出行动，并与环境交互，环境对行为的反应通常可以使用 Simulink 来模拟。
- 在 Matlab 中，Agent、Environment、Observations、Actions 都被表示为变量。



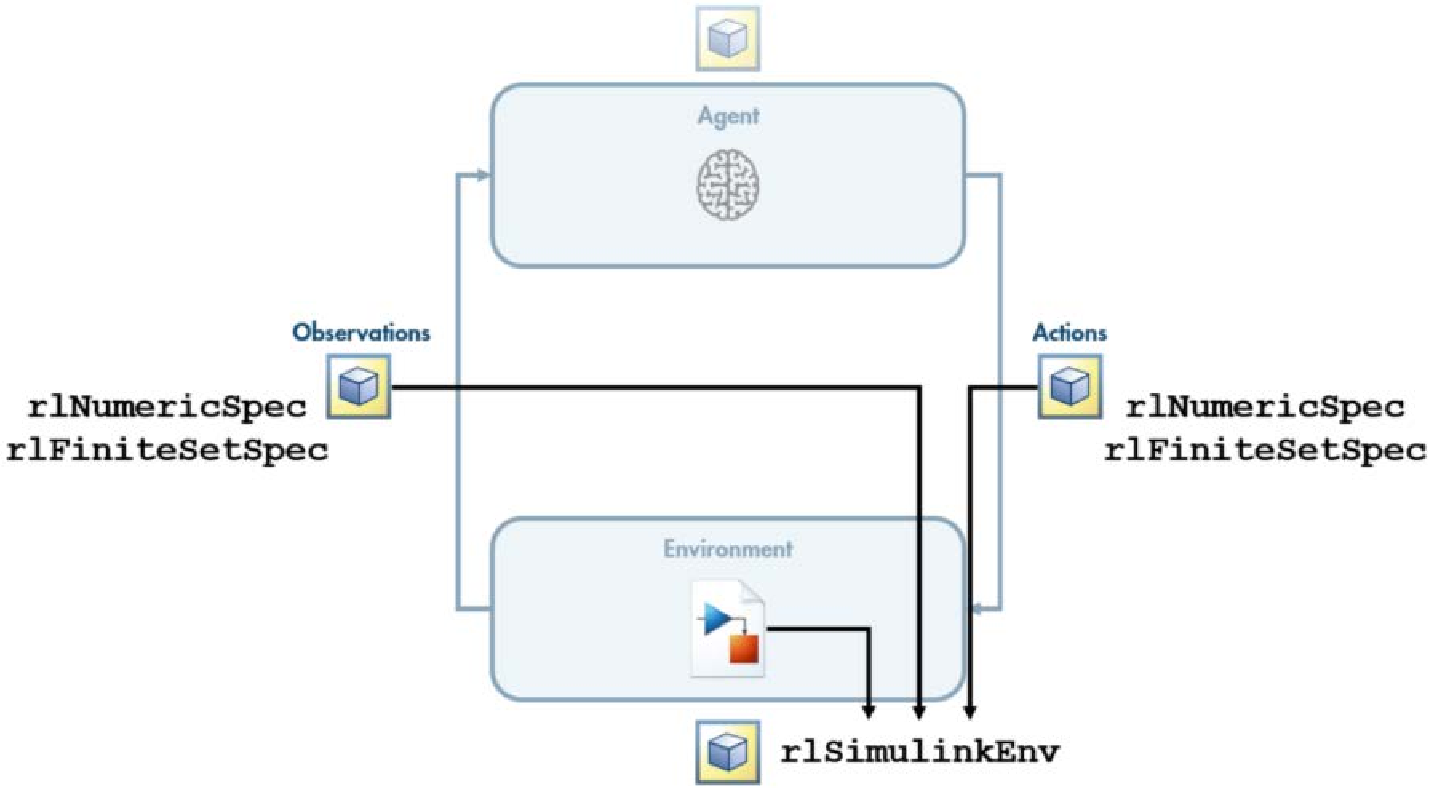
定义环境

- 使用 *rlNumericSpec* 或者 *rlFiniteSetSpec* 声明 Observations 和 Actions 变量



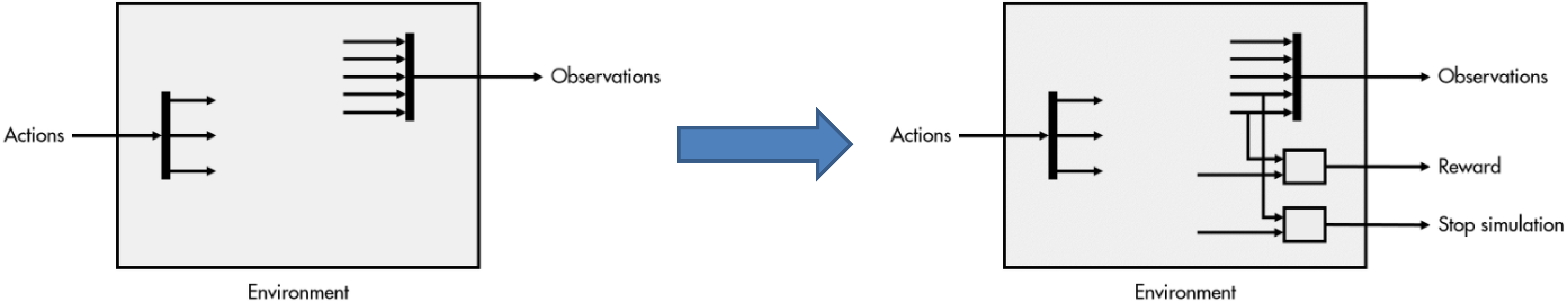
定义环境

■ 使用 *rlSimulinkEnv* 定义一个基于 Simulink 模型的环境变量



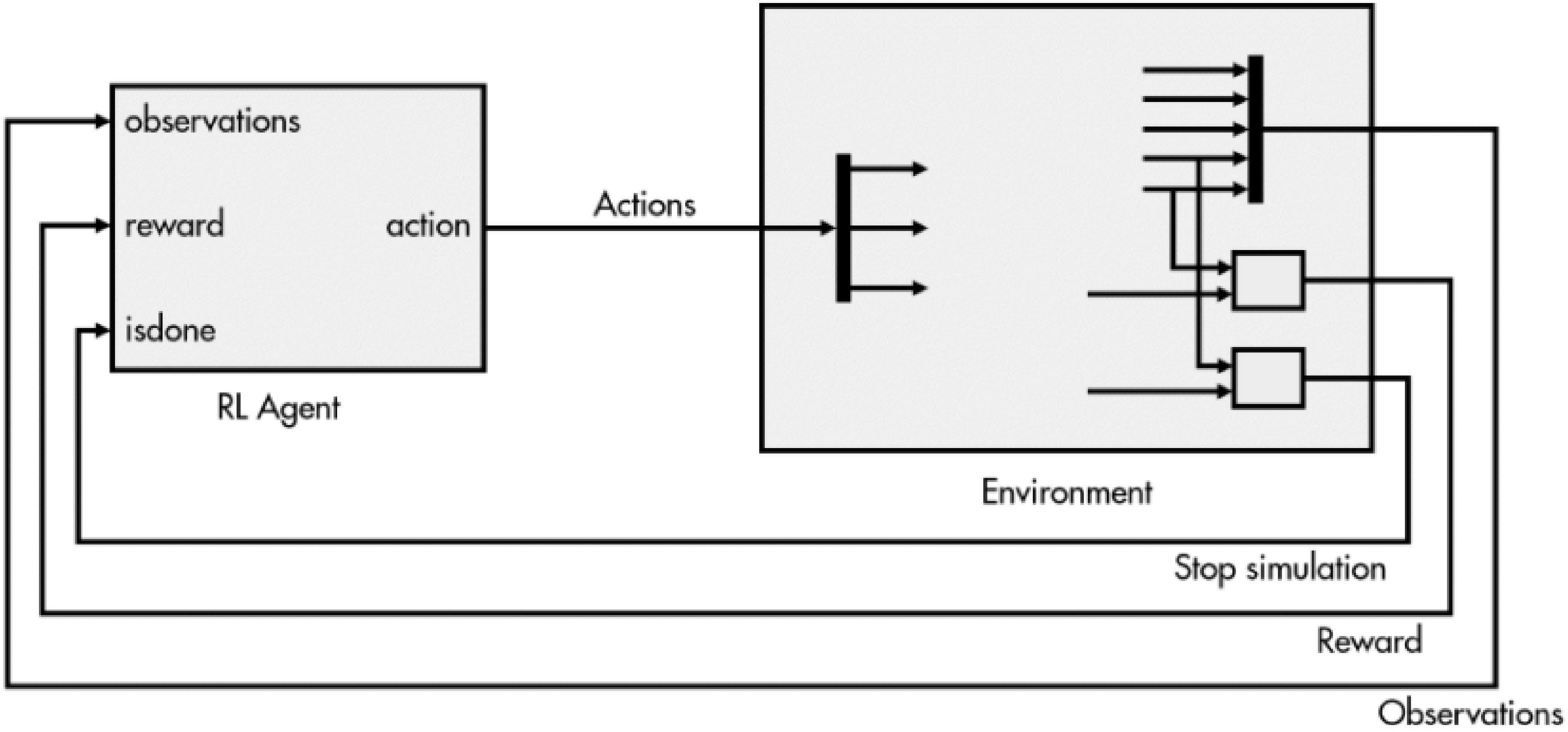
定义环境

- 环境还需要根据 Observations 或者其他信息为 Agent 提供 Reward，同时判定一次模拟过程是否结束



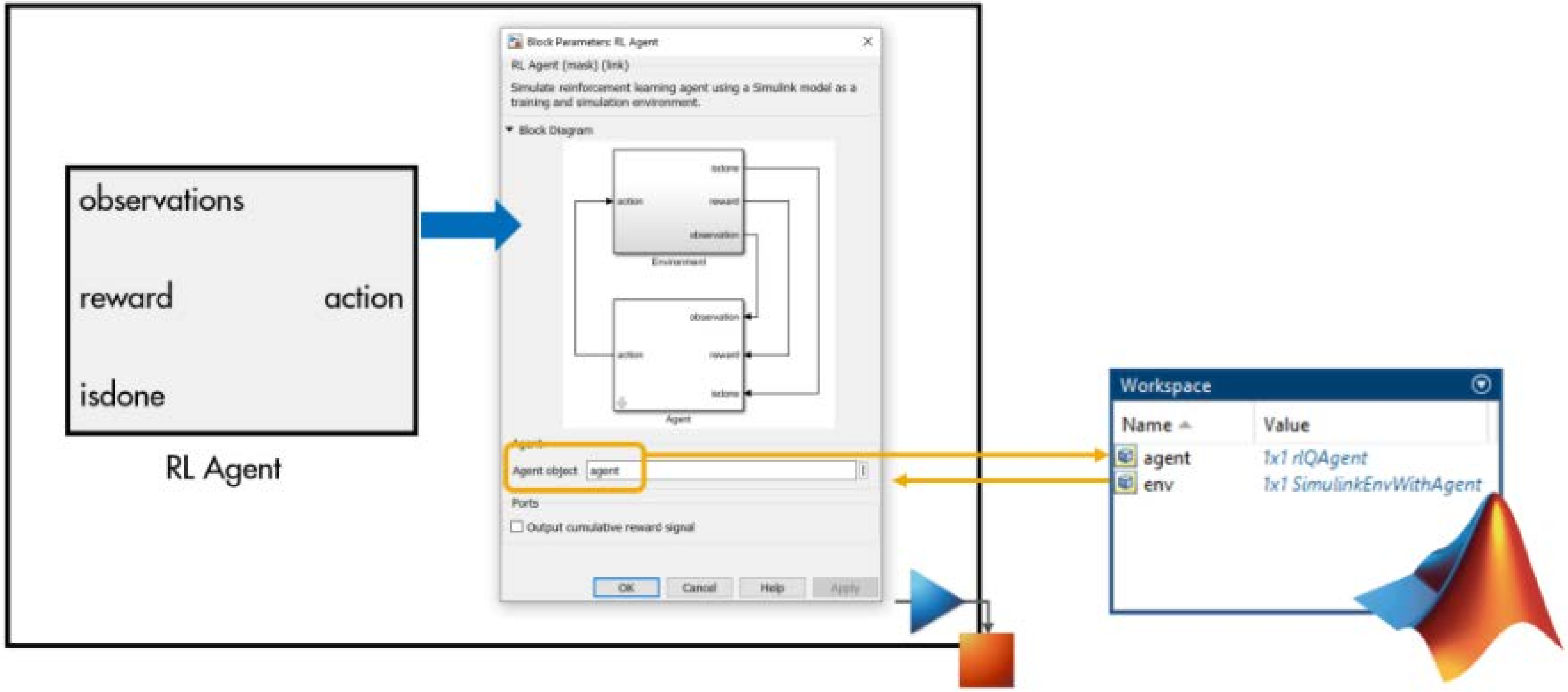
定义环境

■ 在 Simulink 中添加一个 *RL Agent* 模块，并且连接相应输入输出



定义环境

- *RL Agent* 模块需要指明参数 *Agent Object*， 即 Agent 变量名，在训练之前或模拟之前该变量需存在于 Matlab 工作区



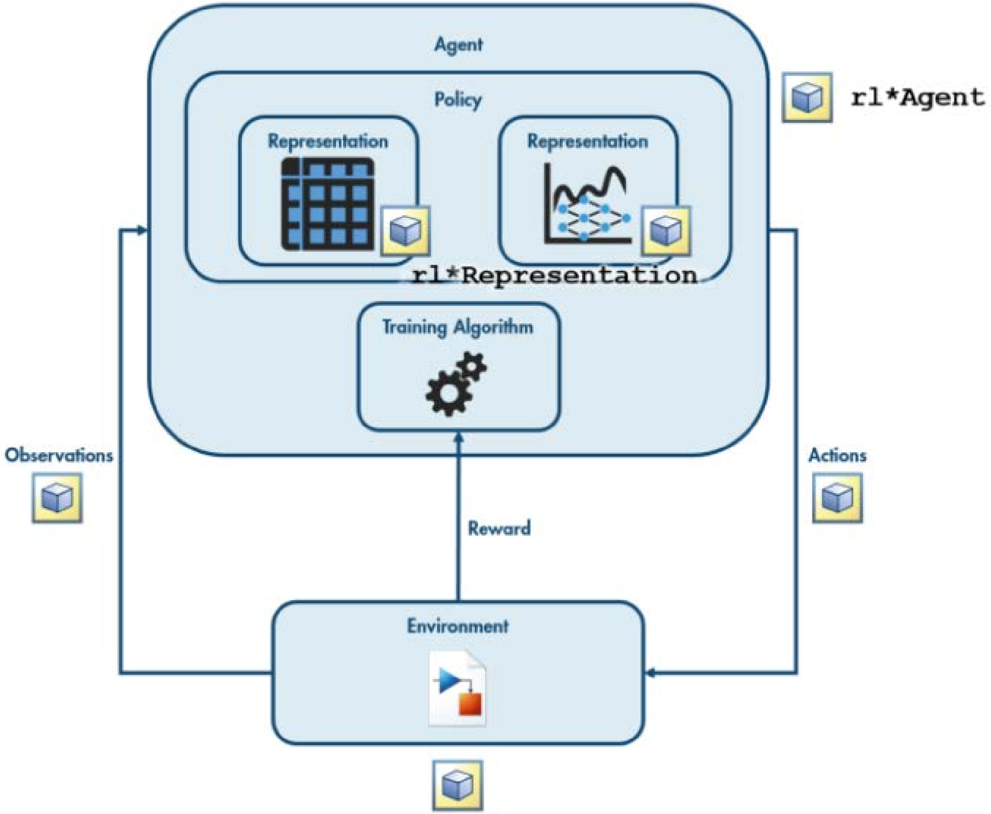
定义 Agent

■ 使用 *rl*Agent* 定义不同学习算法的 Agent，不同类型的 Agent 可能接受不同类型的 Actions

Agent	Type	Action Space
Q-Learning Agents (Q)	Value-Based	Discrete
Deep Q-Network Agents(DQN)	Value-Based	Discrete
SARSA Agents	Value-Based	Discrete
Policy Gradient Agents (PG)	Policy-Based	Discrete or continuous
Actor-Critic Agents (AC)	Actor-Critic	Discrete or continuous
Proximal Policy Optimization Agents (PPO)	Actor-Critic	Discrete or continuous
Deep Deterministic Policy Gradient Agents (DDPG)	Actor-Critic	Continuous
Twin-Delayed Deep Deterministic Policy Gradient Agents (TD3)	Actor-Critic	Continuous
Soft Actor-Critic Agents (SAC)	Actor-Critic	Continuous

定义 Agent

- 使用 *rl*Representation* 定义 Agent 使用的不同策略形式，如 *rlValueRepresentation*, *rlQValueRepresentation*, *rlDeterministicActorRepresentation*, *rlStochasticActorRepresentation*



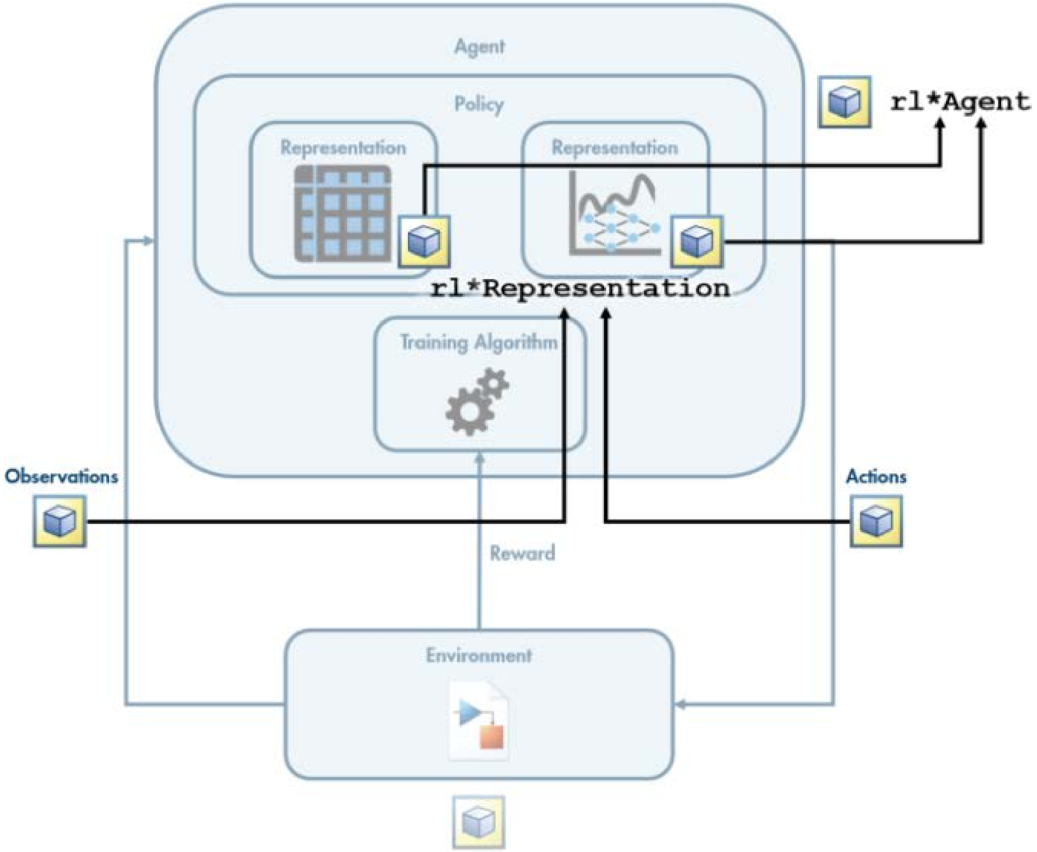
定义 Agent

- 使用 *rl*Representation* 定义 Agent 使用的不同策略形式，不同类型的 Agent 可能需要使用不同的策略形式

Representation	Q, DQN, SARSA	PG	AC, PPO	SAC	DDPG, TD3
Value function critic *V(S)*, which you create using `rlValueRepresentation`		X (if baseline is used)	X		
Q-value function critic *Q(S,A)*, which you create using `rlQValueRepresentation`	X			X	X
Deterministic policy actor *π(S)*, which you create using `rlDeterministicActorRepresentation`					X
Stochastic policy actor *π(S)*, which you create using `rlStochasticActorRepresentation`		X	X	X	

定义 Agent

- 使用定义好的 Observations、Actions 变量定义 Policy， 然后使用 Policy 变量定义 Agent

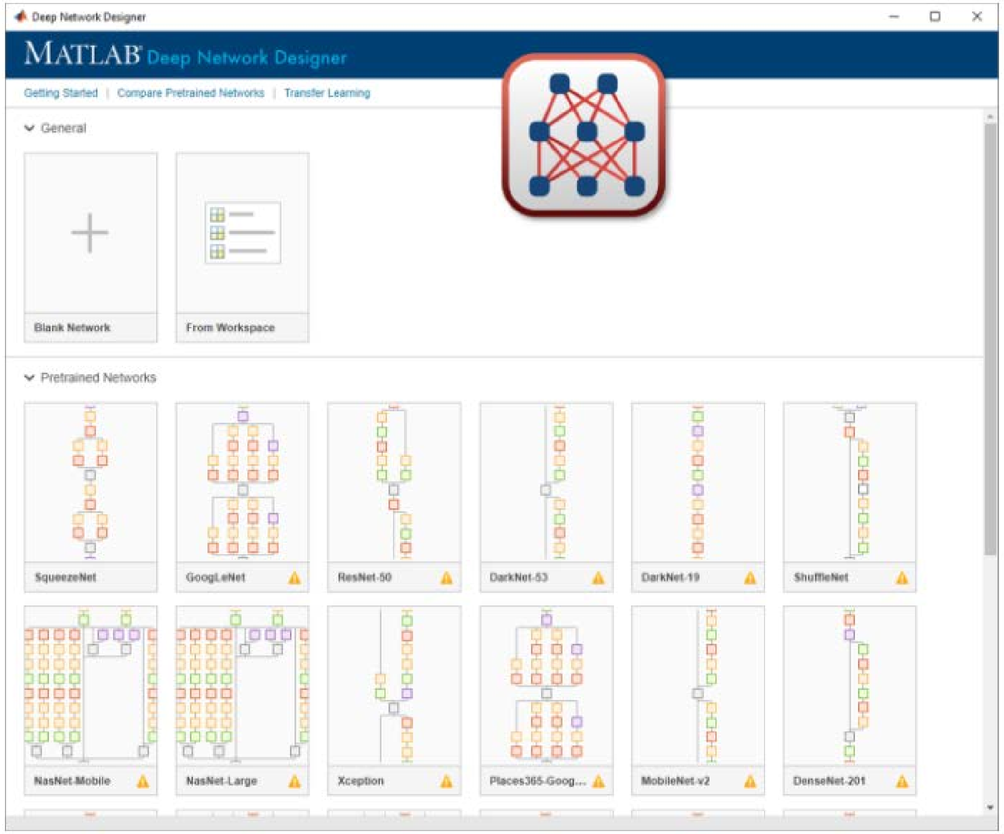


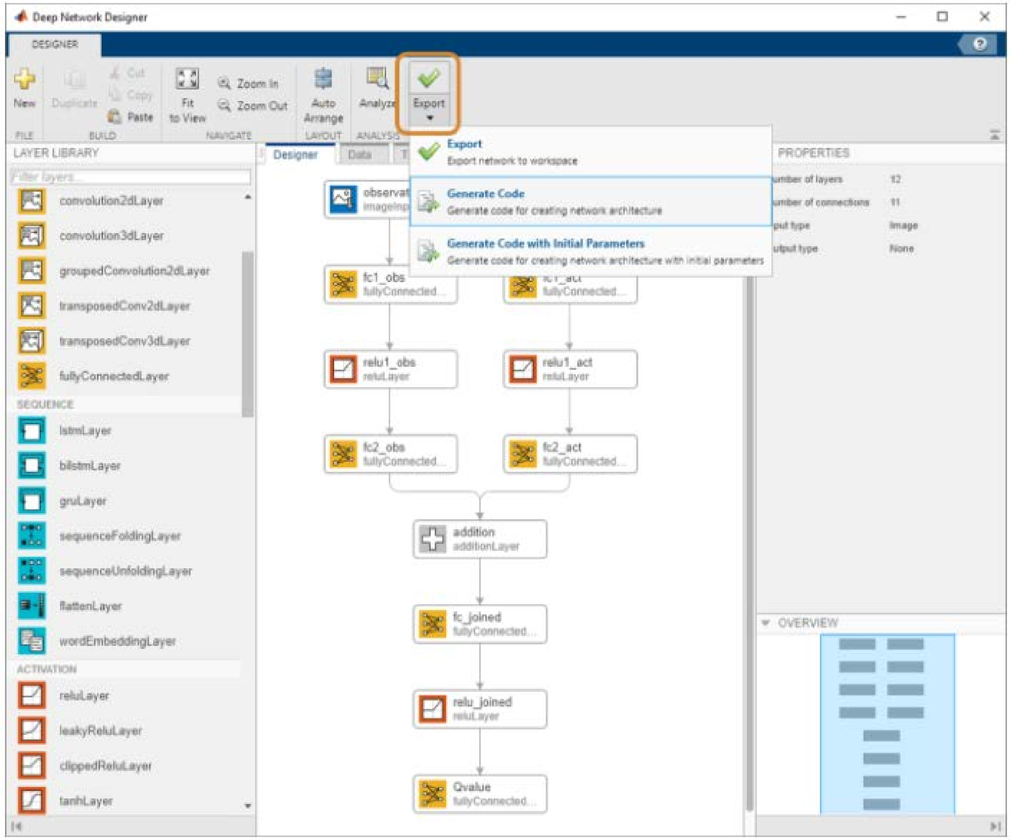
使用 Matlab 进行强化学习

Agent	Actions	Type	Representation(s)
Q-Learning rlQAgent	Discrete	Critic	rlQValueRepresentation
SARSA rlSARSAAgent	Discrete	Critic	rlQValueRepresentation
Deep Q-Network (DQN) rlDQNAgent	Discrete	Critic	rlQValueRepresentation
Policy Gradient rlPGAgent	Discrete or continuous	Actor or Actor-Critic	rlStochasticActorRepresentation (actor) rlValueRepresentation (critic)
Actor-Critic rlACAgent	Discrete or continuous	Actor-Critic	rlStochasticActorRepresentation (actor) rlValueRepresentation (critic)
Deep Deterministic Policy Gradient (DDPG) rlDDPGAgent	Continuous	Actor-Critic	rlDeterministicActorRepresentation (actor) rlQValueRepresentation (critic)
Proximal Policy Optimization (PPO) rlPPOAgent	Discrete or continuous	Actor-Critic	rlStochasticActorRepresentation (actor) rlValueRepresentation (critic)
Twin-Delayed Deep Deterministic Policy Gradient (TD3) rlTD3Agent	Continuous	Actor-Critic	rlDeterministicActorRepresentation (actor) rlQValueRepresentation (critic)
Soft Actor-Critic (SAC) rlSACAgent	Continuous	Actor-Critic	rlStochasticActorRepresentation (actor) rlQValueRepresentation (critic)

定义 Agent

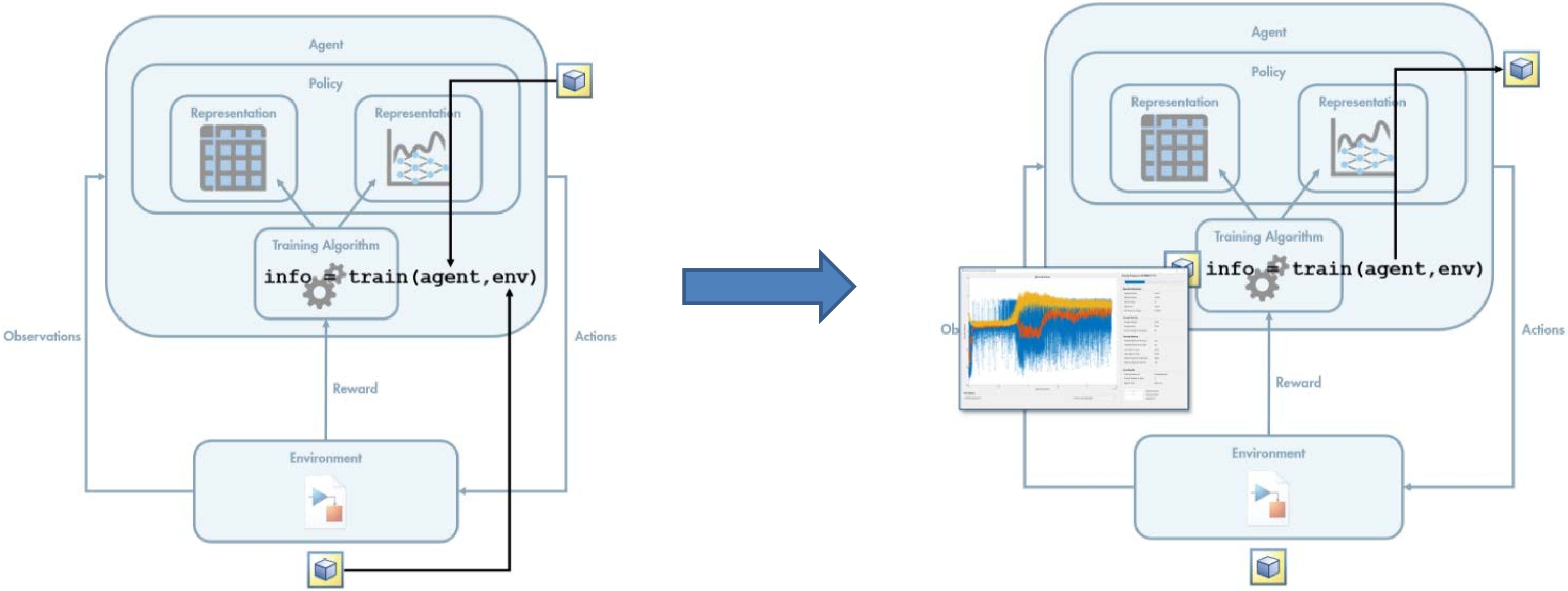
- Deep Network Designer





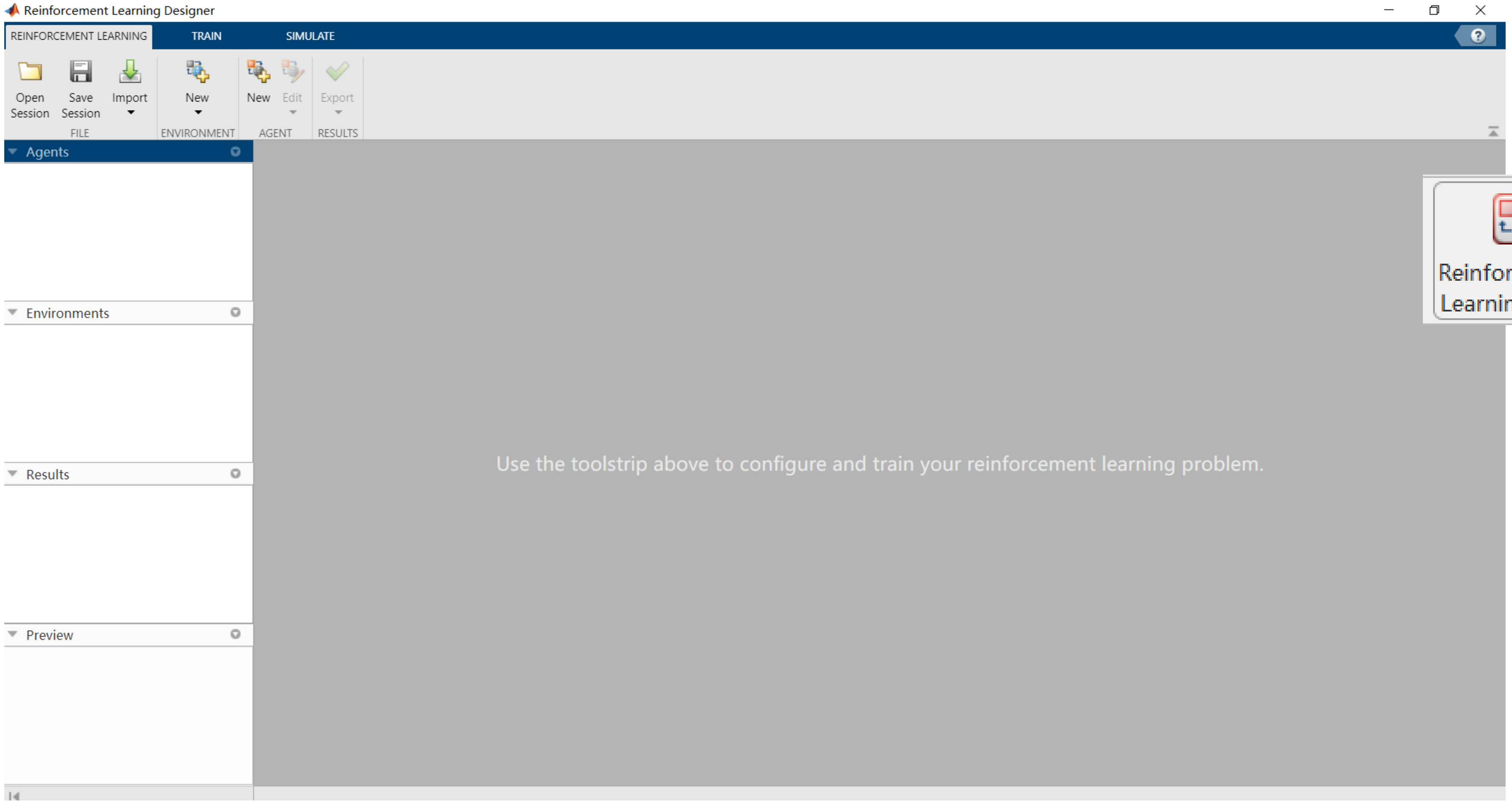
训练

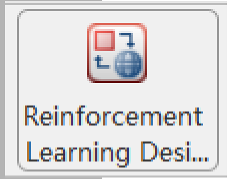
- 定义好环境和 Agent 之后，传入 *train* 进行训练，Agent 在训练过程中直接会被改变，输出为训练过程中产生的信息。训练结束以后可以使用 sim 进行模拟验证。



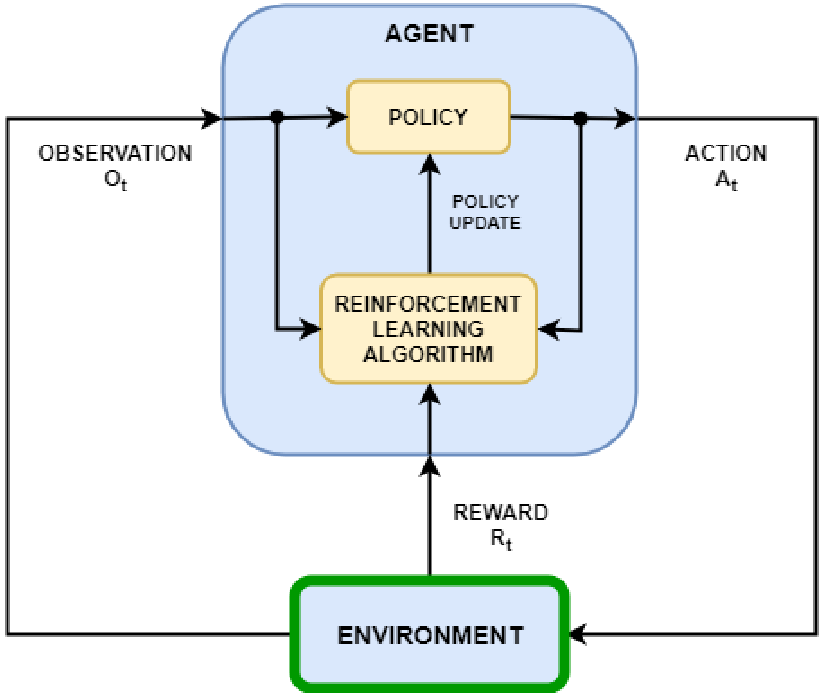
训练

■ Reinforcement Learning Designer



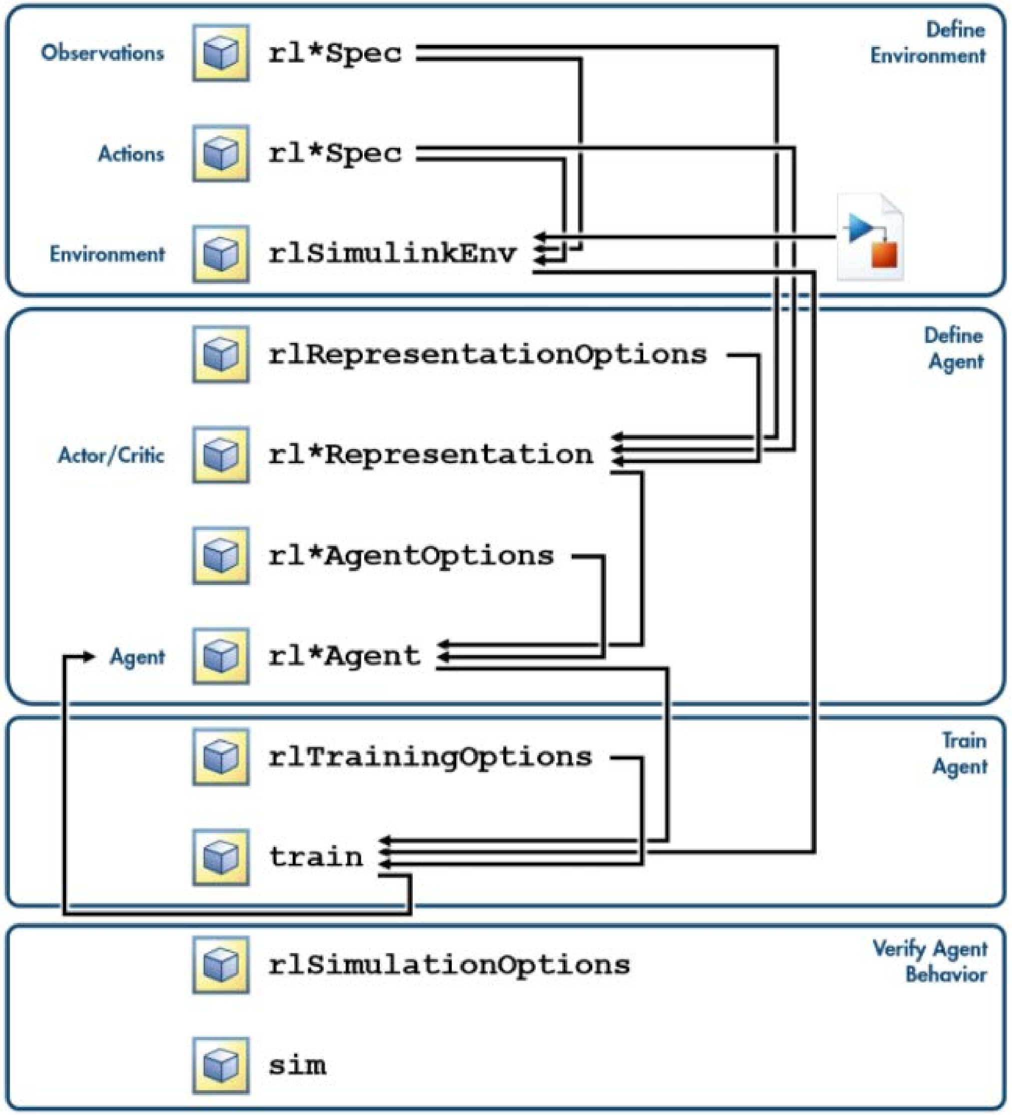


总结


AGENT
POLICY
OBSERVATION
O_t
ACTION
A_t
POLICY
UPDATE
REINFORCEMENT
LEARNING
ALGORITHM
REWARD
R_t
ENVIRONMENT


Define Environment
Set Agent Options
Define Agent
Train Agent
Evaluate Agent

总结

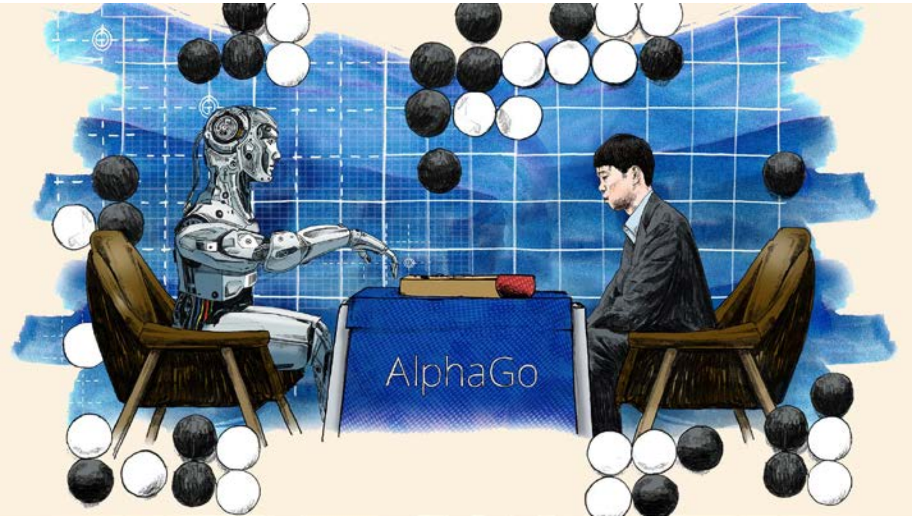

Observations
rl*Spec
Actions
rl*Spec
Environment
rlSimulinkEnv
Define Environment
rlRepresentationOptions
Actor/Critic
rl*Representation
rl*AgentOptions
Agent
rl*Agent
Define Agent
rlTrainingOptions
train
Train Agent
rlSimulationOptions
sim
Verify Agent Behavior

总结

	Functions	Options
Interface	rlFiniteSetSpec rlNumericSpec	
Environment	rlSimulinkEnv	
Representations (Actors & Critics)	rlQValueRepresentation rlValueRepresentation rlDeterministicActorRepresentation rlStochasticActorRepresentation	rlRepresentationOptions
Agents	rlQAgent rlSARSAAgent rlDQNAgent rlPGAgent rlACAgent rlPPOAgent rlDDPGAgent rlTD3Agent rlSACAgent	rlQAgentOptions rlSARSAAgentOptions rlDQNAgentOptions rlPGAgentOptions rlACAgentOptions rlPPOAgentOptions rlDDPGAgentOptions rlTD3AgentOptions rlSACAgentOptions
Training	train	rlTrainingOptions
Simulation	sim	rlSimulationOptions

Matlab 中的强化学习

- 回顾强化学习基础知识，了解强化学习的基本工作流程
- 学习如何在 Matlab 中进行强化学习
- demo 讲解

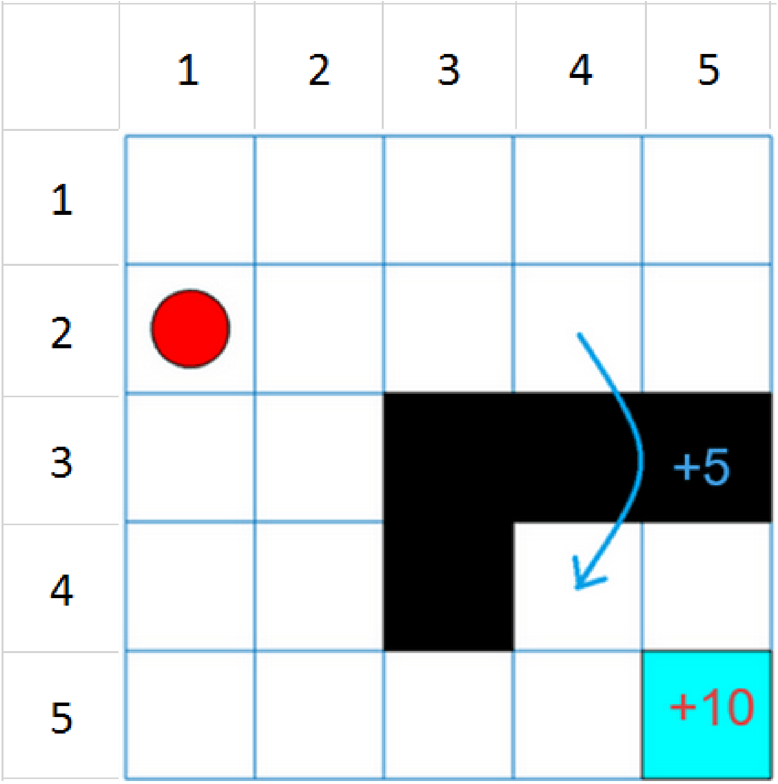


基础网格系统（Basic Grid World）

■ Agent 的目标是尽可能获得最大的分数

1. The grid world is 5-by-5 and bounded by borders, with four possible actions (North = 1, South = 2, East = 3, West = 4).
2. The agent begins from cell [2,1] (second row, first column).
3. The agent receives a reward +10 if it reaches the terminal state at cell [5,5] (blue).
4. The environment contains a special jump from cell [2,4] to cell [4,4] with a reward of +5.
5. The agent is blocked by obstacles (black cells).
6. All other actions result in –1 reward.

```
env = rlPredefinedEnv("BasicGridWorld");
```



基础网格系统（Actions & Observations）

■ Agent 可以产生往东、南、西、北四个方向走的动作

(North = 1, South = 2, East = 3, West = 4)

```
actionInfo =
  rlFiniteSetSpec - 属性:

       Elements: [4×1 double]
           Name: "MDP Actions"
    Description: [0×0 string]
      Dimension: [1 1]
       DataType: "double"
```

■ Agent 可以观测到方块位于 25 个网格中的哪一个网格

```
obsInfo =
  rlFiniteSetSpec - 属性:

       Elements: [25×1 double]
           Name: "MDP Observations"
    Description: [0×0 string]
      Dimension: [1 1]
       DataType: "double"
```

基础网格系统（Reward）

- Reward 由事先规定好的规则给出

3. The agent receives a reward +10 if it reaches the terminal state at cell [5,5] (blue).
4. The environment contains a special jump from cell [2,4] to cell [4,4] with a reward of +5.
5. The agent is blocked by obstacles (black cells).
6. All other actions result in −1 reward.

基础网格系统（Agent）

- 场景较简单，状态、动作数量不多，采用 QTable 的策略形式即可

```
%% define agent
qTable = rlTable(getObservationInfo(env),getActionInfo(env));
qRepresentation = rlQValueRepresentation(qTable,getObservationInfo(env),getAc
tionInfo(env));
qRepresentation.Options.LearnRate = 1;
agentOpts = rlQAgentOptions;
agentOpts.EpsilonGreedyExploration.Epsilon = .04;
qAgent = rlQAgent(qRepresentation,agentOpts);
```

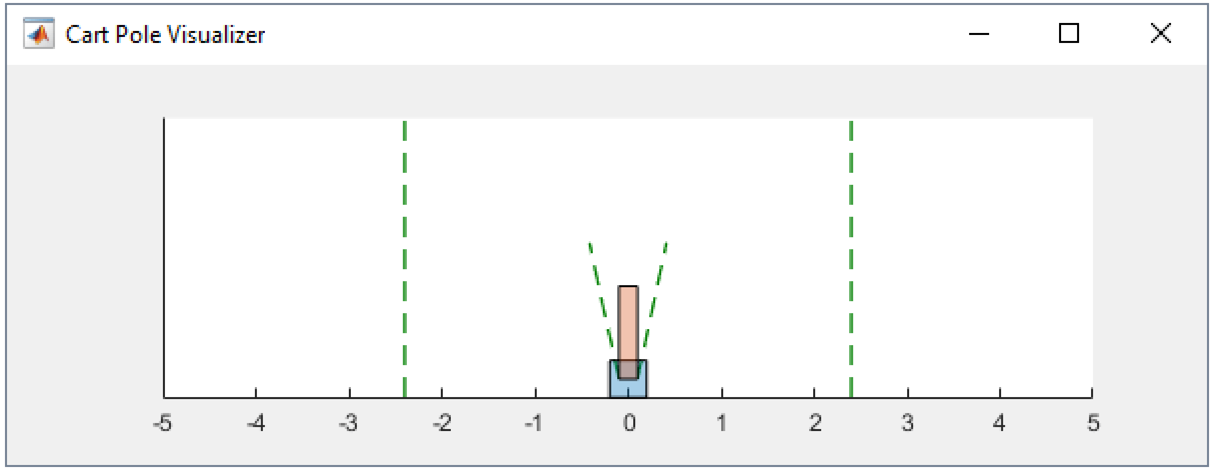
基础网格系统（训练）

```
%% train agent
trainOpts = rlTrainingOptions;
trainOpts.MaxStepsPerEpisode = 50;
trainOpts.MaxEpisodes= 200;
trainOpts.StopTrainingCriteria = "AverageReward";
trainOpts.StopTrainingValue = 11;
trainOpts.ScoreAveragingWindowLength = 30;
trainingStats = train(qAgent,env,trainOpts);
```

车杆系统（Cart-Pole）

- Agent 的目标是通过在移动的小车上施加水平方向上的力来保持杆的平衡
- 平衡条件 1： 杆的角度保持在给定范围内
- 平衡条件2： 小车的位置保持在给定范围内

```
env = rlPredefinedEnv("CartPole-Discrete");
```



车杆系统（环境）

Environment Properties

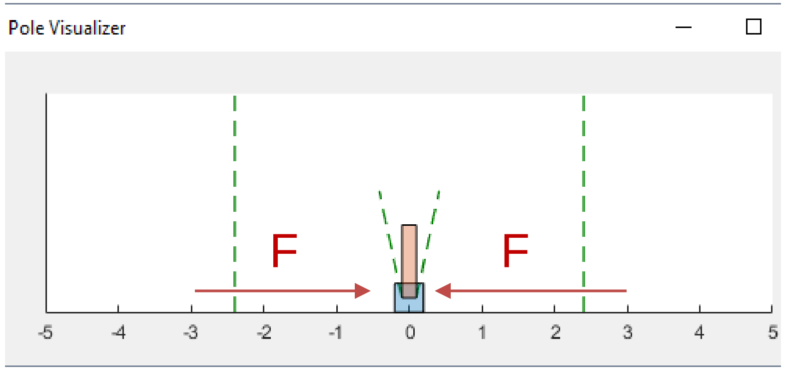
Property	Description	Default
`Gravity`	Acceleration due to gravity in meters per second	`9.8`
`MassCart`	Mass of the cart in kilograms	`1`
`MassPole`	Mass of the pole in kilograms	`0.1`
`Length`	Half the length of the pole in meters	`0.5`
`MaxForce`	Maximum horizontal force magnitude in newtons	`10`
`Ts`	Sample time in seconds	`0.02`
`ThetaThresholdRadians`	Pole angle threshold in radians	`0.2094`
`XThreshold`	Cart position threshold in meters	`2.4`
`RewardForNotFalling`	Reward for each time step the pole is balanced	`1`
`PenaltyForFalling`	Reward penalty for failing to balance the pole	Discrete — `-5` Continuous — `-50`
`State`	Environment state, specified as a column vector with the following state variables: • Cart position • Derivative of cart position • Pole angle • Derivative of pole angle	`[0 0 0 0]'`

车杆系统（Actions）

■ Agent 通过施加一个水平方向上的力与环境进行交互

```
actInfo =
  rlFiniteSetSpec - 属性:

      Elements: [-10 10]
          Name: "CartPole Action"
   Description: [0×0 string]
     Dimension: [1 1]
      DataType: "double"
```

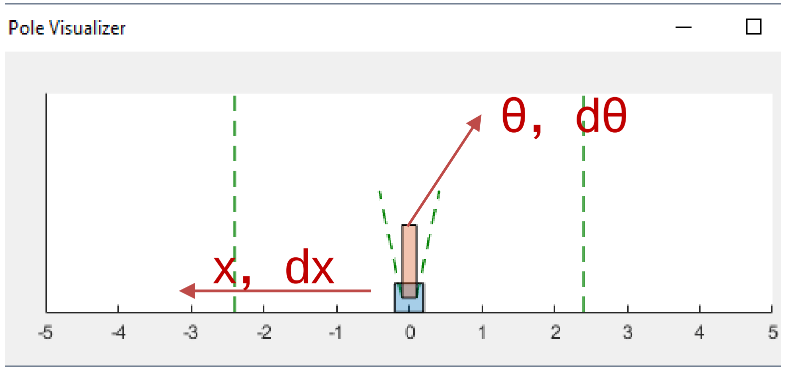


车杆系统（Observations）

- Agent 能够获取环境的所有状态值

```
obsInfo =
  rlNumericSpec - 属性:

    LowerLimit: -Inf
    UpperLimit: Inf
          Name: "CartPole States"
   Description: "x, dx, theta, dtheta"
     Dimension: [4 1]
      DataType: "double"
```



车杆系统（Reward）

- Reward 信号由两个部分组成：
- Positive reward：在一个步长下，杆的偏移没有超过边界
- Negative penalty：在一个步长下，杆或者车的偏移超过边界

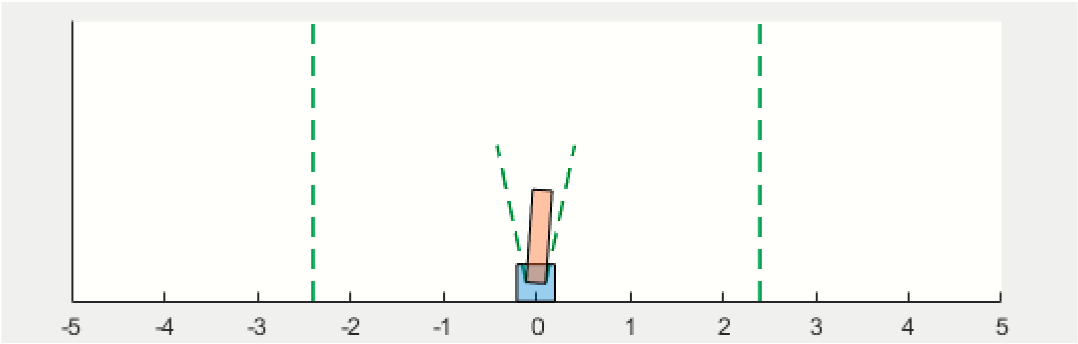
`RewardForNotFalling`	Reward for each time step the pole is balanced	`1`
`PenaltyForFalling`	Reward penalty for failing to balance the pole	Discrete — `-5` Continuous — `-50`

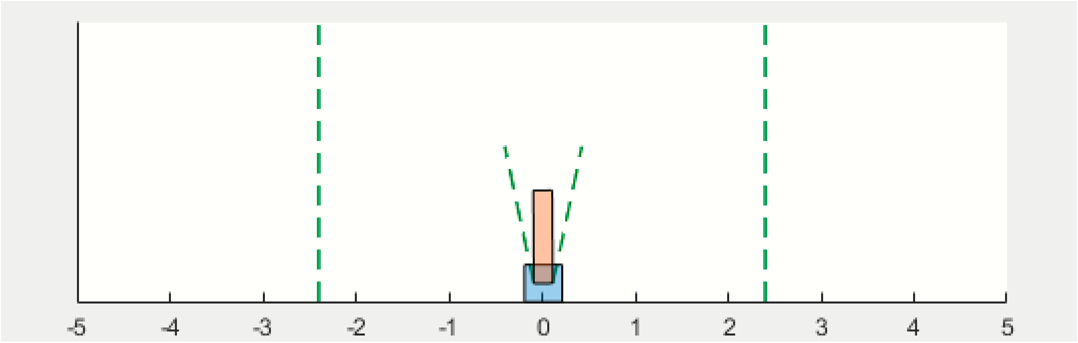
车杆系统（Agent）

```
%% critic
criticNetwork = [
    featureInputLayer(4,'Normalization','none','Name','state')
    fullyConnectedLayer(1,'Name','CriticFC')];
criticOpts = rlRepresentationOptions('LearnRate',8e-3,'GradientThreshold',1);
critic = rlValueRepresentation(criticNetwork,obsInfo,'Observation',{'state'},criticOpts);
%% actor
actorNetwork = [
    featureInputLayer(4,'Normalization','none','Name','state')
    fullyConnectedLayer(2,'Name','fc')
    softmaxLayer('Name','actionProb')];
actorOpts = rlRepresentationOptions('LearnRate',8e-3,'GradientThreshold',1);
actor = rlStochasticActorRepresentation(actorNetwork,obsInfo,actInfo,...
    'Observation',{'state'},actorOpts);
%% agent
agentOpts = rlACAgentOptions(...
    'NumStepsToLookAhead',32, ...
    'DiscountFactor',0.99);
agent = rlACAgent(actor,critic,agentOpts);
```

车杆系统（训练）

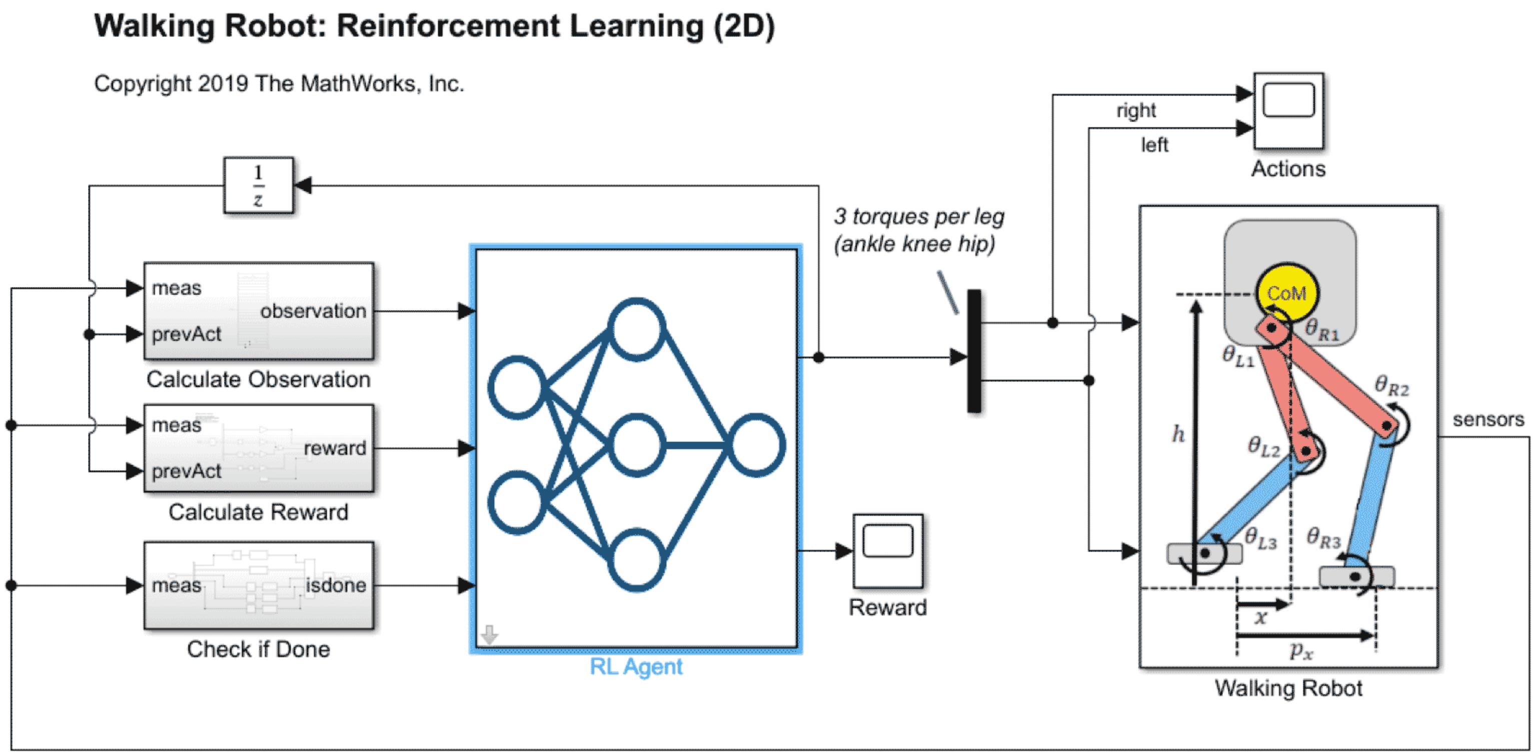
```
%% Train the agent
trainOpts = rlTrainingOptions(...
    'MaxEpisodes',1000,...
    'MaxStepsPerEpisode',500,...
    'Verbose',false,...
    'Plots','training-progress',...
    'StopTrainingCriteria','AverageReward',...
    'StopTrainingValue',480,...
    'ScoreAveragingWindowLength',10);
plot(env)
trainingStats = train(agent,env,trainOpts);
```



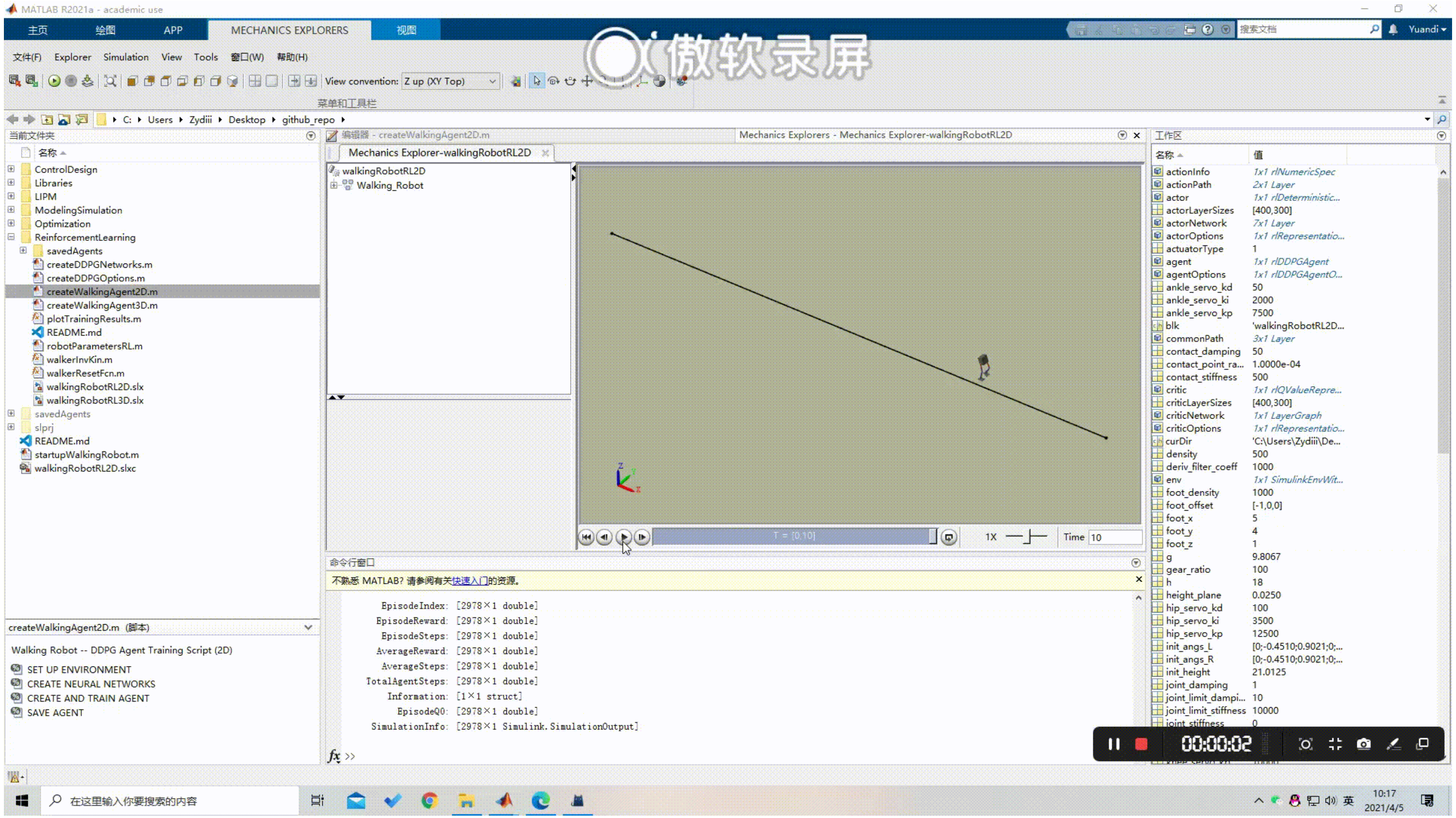


拓展——行走机器人（Walking Robot）

■ Agent 的目标是机器人能够保持身体平衡，在行走过程中不摔倒，同时移动速度尽可能快



拓展——行走机器人（Walking Robot）



学习资料

- Deep Reinforcement Learning for Walking Robots
  https://www.mathworks.com/videos/deep-reinforcement-learning-for-walking-robots--1551449152203.html
- Reinforcement Learning Onramp
  https://www.mathworks.com/learn/tutorials/reinforcement-learning-onramp.html
- Reinforcement Learning Toolbox
  https://www.mathworks.com/products/reinforcement-learning.html
- Reinforcement Learning: A Brief Guide
  https://ww2.mathworks.cn/company/newsletters/articles/reinforcement-learning-a-brief-guide.html?s_tid=srchtitle
- Reinforcement Learning Toolbox Help
  https://ww2.mathworks.cn/help/reinforcement-learning/

Question & Answer

任何疑问和建议，请不要犹豫！

王　赓：wgeng@sjtu.edu.cn