教材《机器学习与大语言模型》配套电子教案

机器学习

第十七讲: 强化学习

顾小东
上海交通大学计算机学院

What we have learned so far?

监督学习

数据: (x, y)
x表示数据, y表示标签

学习目标: 学习函数映射
$x \rightarrow y$

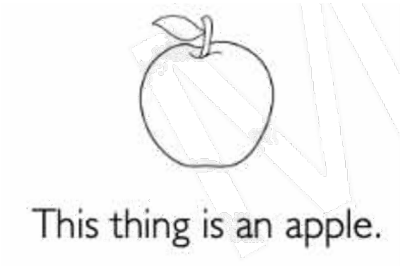
典型任务: 分类、回归、目标检测、语义分割等

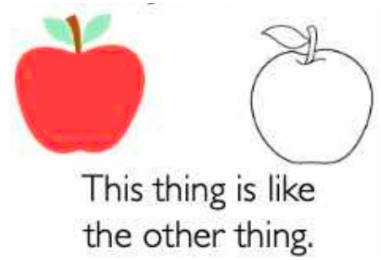
无监督学习

数据: x
x表示数据, 无标签

学习目标: 学习数据隐含或潜在的结构

典型任务: 聚类、降维、概率密度估计等





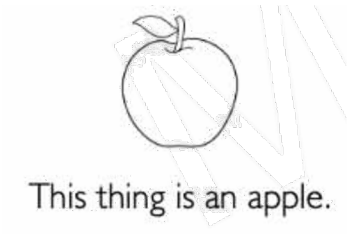
第三类机器学习

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn to map $x \rightarrow y$

Examples: Classification, regression, etc.

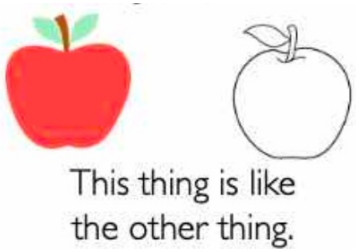


Unsupervised Learning

Data: x
x is data, no label

Goal: Learn underline structure.

Examples: Clustering, dim reduction, etc.



Reinforcement Learning

Data: *state-action* pairs + rewards

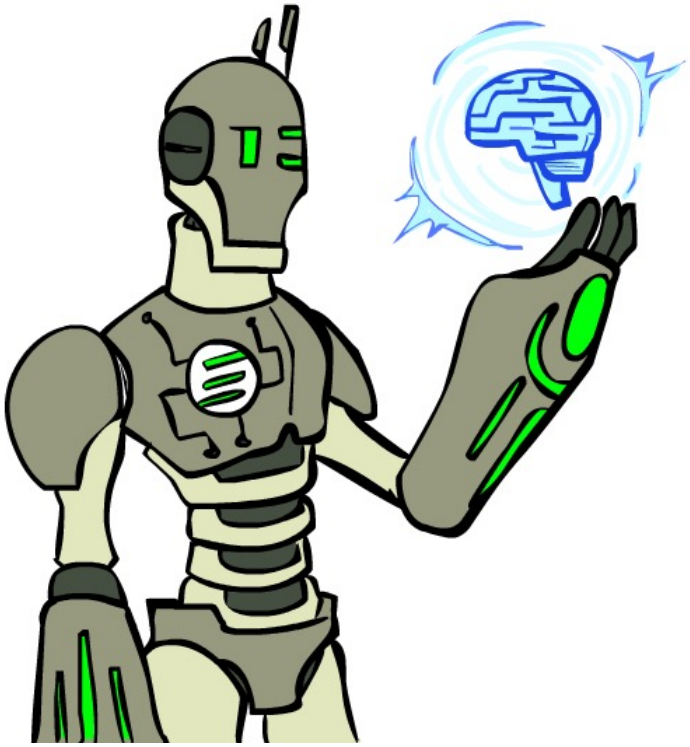
Goal: Obtain maximum reward from environment.

Examples: Robot, Game, etc.

内容提纲

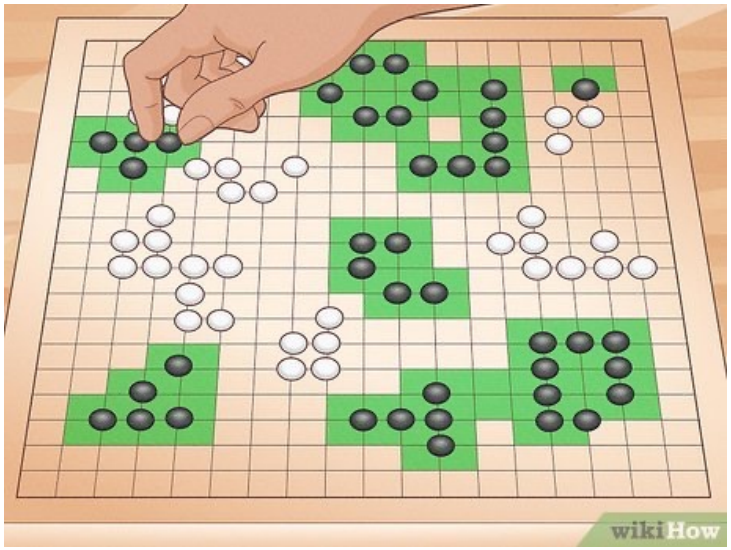
Reinforcement Learning

- 问题建模 (MDP)
- Q Learning
- Deep Q Learning
- Policy Gradient
- Actor-Critic



强化学习的问题设定

Scenario: An AI player wants to learn how to play a game.

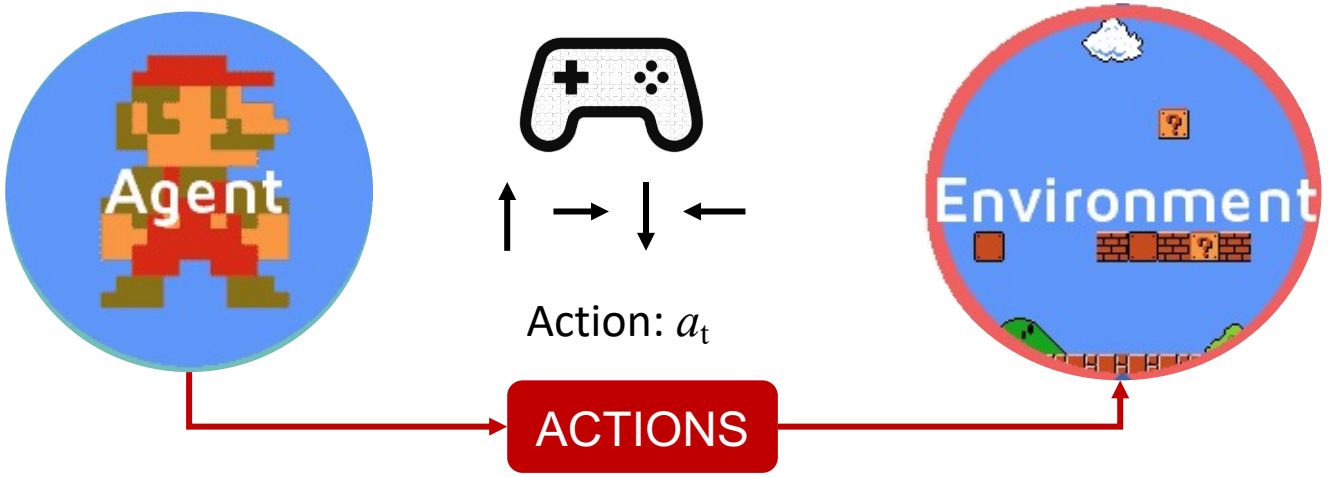


How do I act (move/jump/attack) given different screens?

强化学习的问题设定

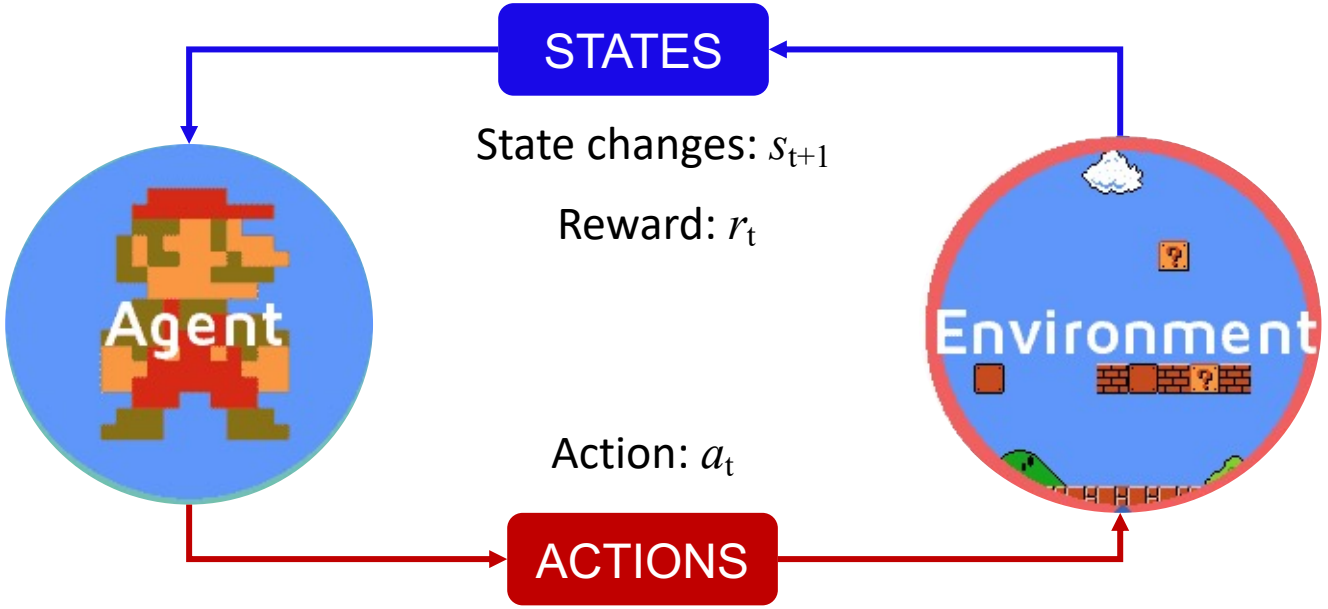
Agent:一个执行动作并且学习知识的扮演者
Environment:智能体所处的外部“世界”，智能体可以对其采取动作

强化学习的问题设定



Action: a move the agent can make in the environment.
Action space A: the set of possible actions an agent can make in the environment.

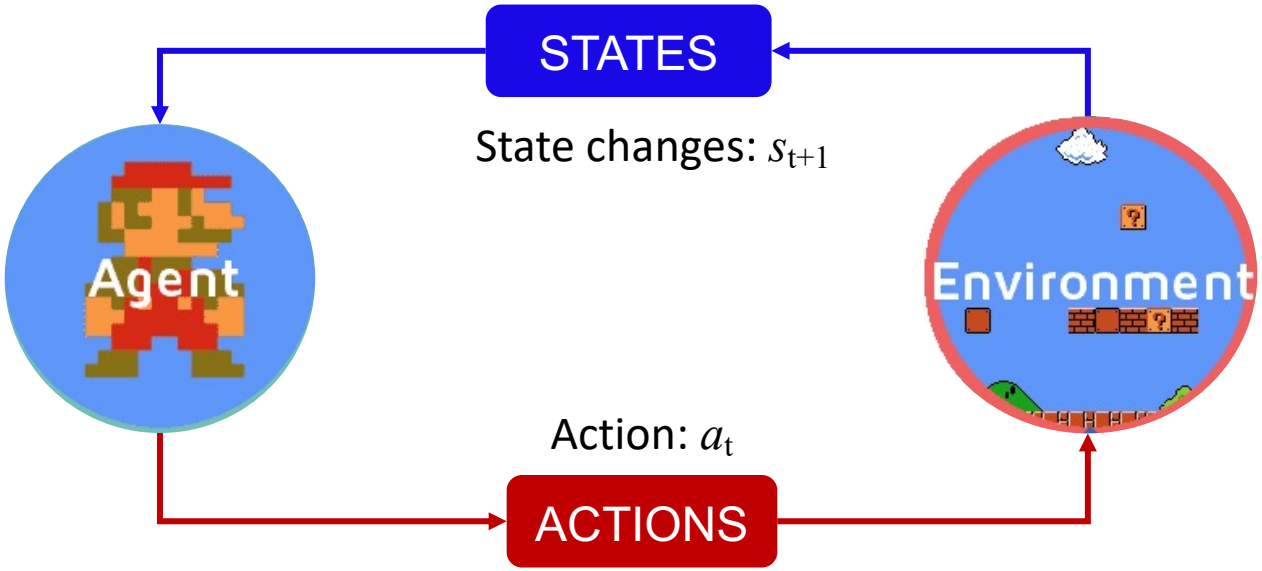
强化学习的问题设定



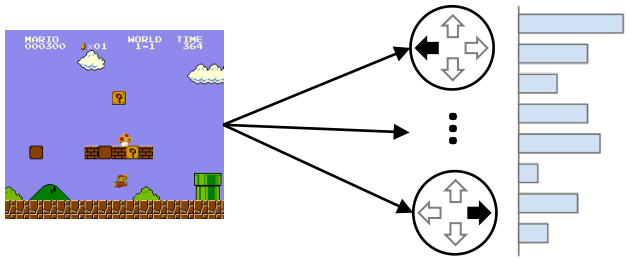
State: observations

Reward: feedback that scores the agent's action.

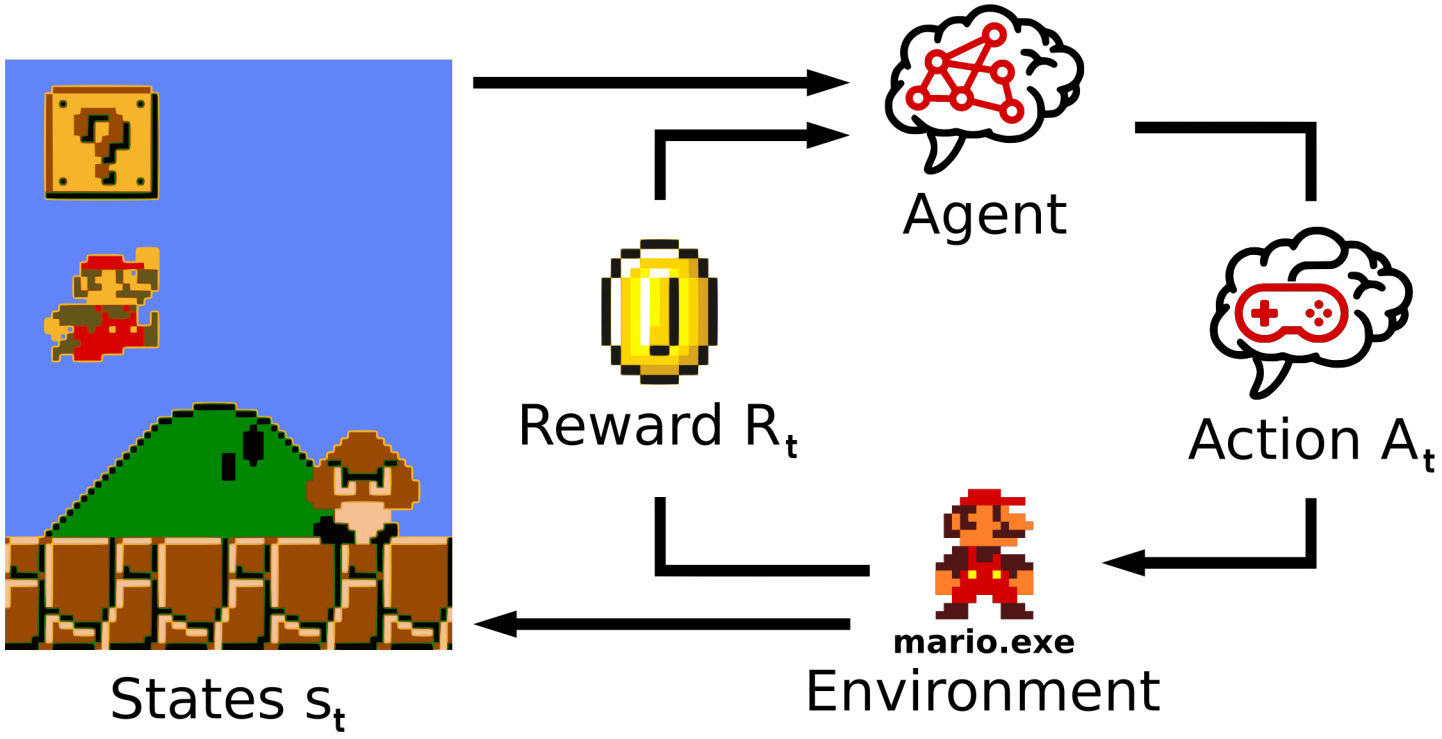
强化学习的问题设定



Policy π: $\mathrm{S} \rightarrow \mathrm{A}$: a function that maps from state to action



完整的问题框架



Thinking: What about other scenarios (e.g., baby, Atari, robotics, go, autonomous driving, etc)?

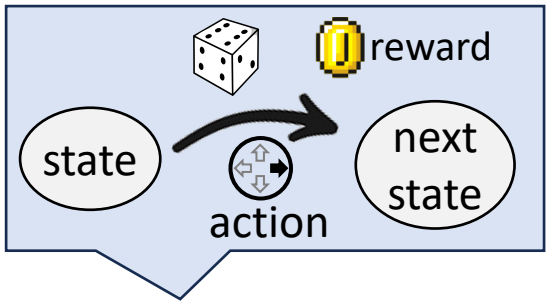
马尔可夫决策过程 (MDP)

- 马尔科夫决策过程是在马尔科夫过程基础上引入动作和奖励的随机过程

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

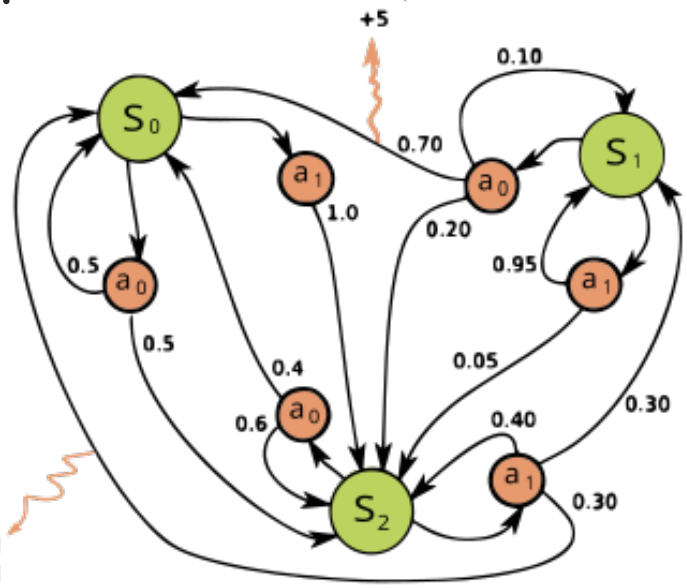
⇩

MDP: $\mathbb{P}[S_{t+1}|S_t, A_t]$



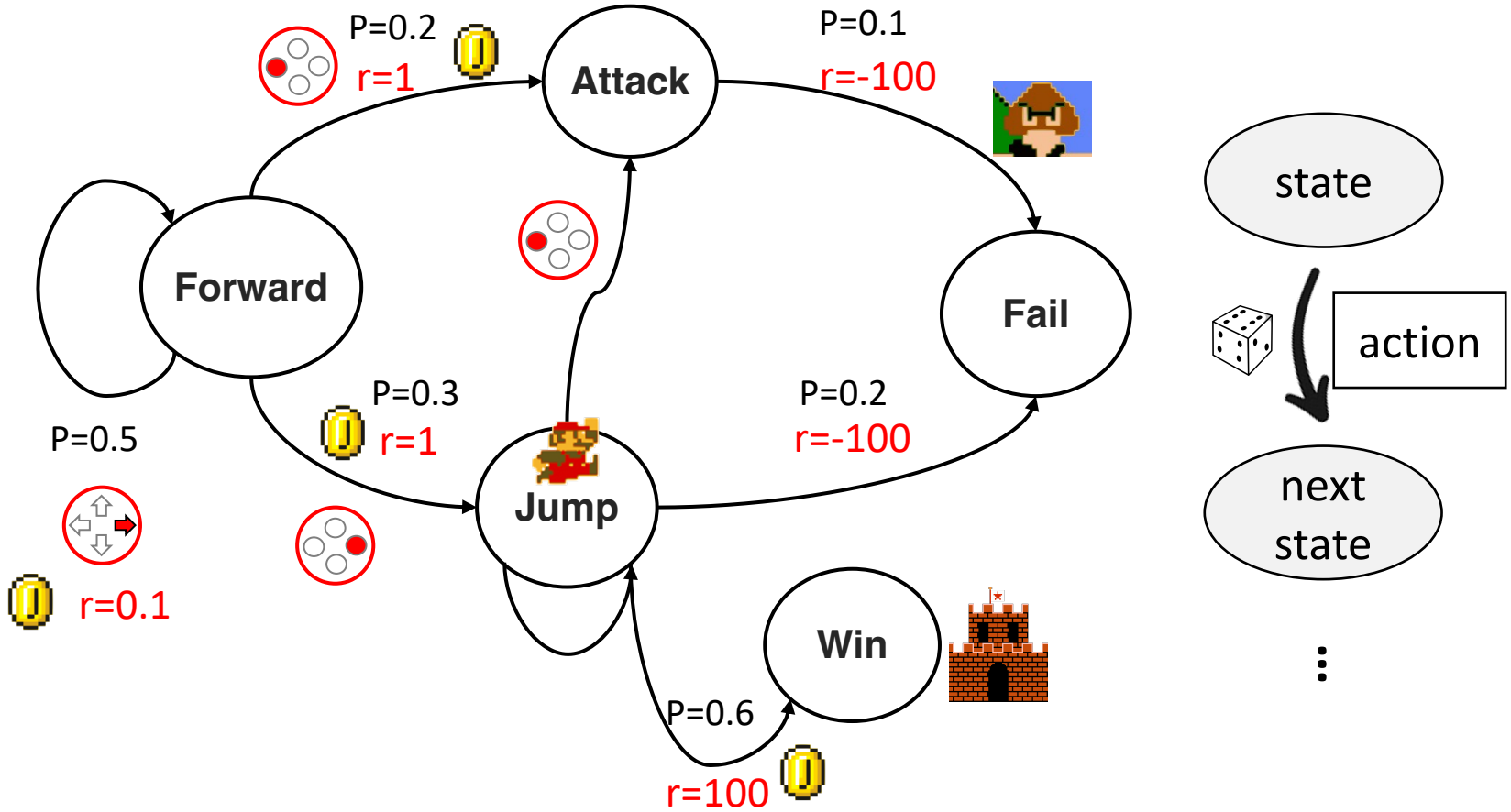
- MDP用五元组表示 (**S**, **A**, $\{\mathbf{P}_{sa}\}$, γ, **R**):

S: a set of states
A: a set of actions (e.g.,← ↑→↓)
$\{\mathbf{P}_{sa}\}$: probabilities of state transition
R: S×A→ R: Reward function
γ∈[0,1]: the discount of future rewards



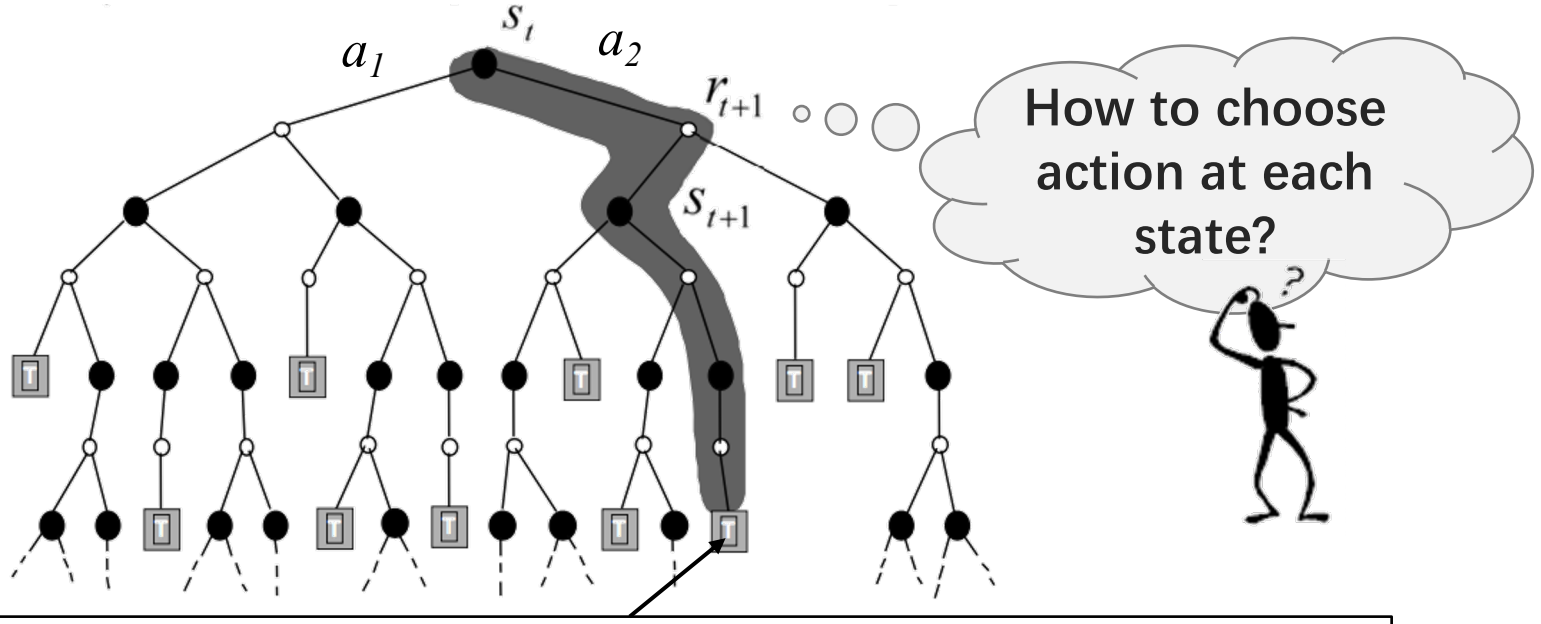
MDP为强化学习建立了一种数学框架

MDP举例


P=0.2
r=1
Attack
P=0.1
r=-100
Forward
Fail
P=0.3
r=1
P=0.2
r=-100
P=0.5
Jump
r=0.1
Win
P=0.6
r=100
state
action
next state

强化学习的学习目标

- The agent finds a path in the **state space** that potentially leads to the maximum long-term return $\boldsymbol{R}_t$.



Actual long-term return following s_t: $\boldsymbol{R}_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + \ldots$

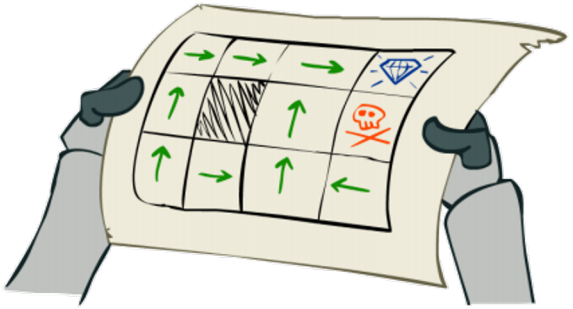
Discounted long-term return: $\boldsymbol{R}_t = \sum_{i=t}^{\infty} \gamma^i r_i = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$

Scoring (S, a) Pairs in MDP: Q-function

- The **Q-function** captures the **expected total future reward** an agent in state s can receive by executing a certain action a

$$Q(s_t, a_t) = \mathbb{E}[R_t \mid s_t, a_t]$$

score(state, action)



- Q-function implicitly defines the optimal **policy π(s)**, for inferring the **best action to take** at state s

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

The policy should choose an action that maximizes future reward.

如何计算Q值？

蒙特卡洛预演

Use real return

时间差分

Use the value from the next state

方案一：蒙特卡洛预演(Rollout)

- 采用最朴素的思想：值(value) = 平均累计奖励(mean return)
- 直接从经验片段学习

$$R_t = r_t + \gamma r_{t+1} \ldots + \gamma^n r_{t+n} + \ldots$$

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t] \approx \frac{1}{N} \sum_1^N R_t$$

期望累计奖励

经验累计奖励

算法:

- 使用策略π从状态s_t采样N个片段,计算每个片段的累计奖励R_t
- 计算平均累计奖励, 更新Q-score

方案二：迭代式估算

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = r_t + \gamma r_{t+1} \ldots + \gamma^n r_{t+n} + \ldots$$

$$\begin{aligned} Q(s_t, a_t) &= \mathbb{E}[R_t | s_t, a_t] \\ &= \mathbb{E}[r_t + \gamma r_{t+1} \ldots + \gamma^n r_{t+n} + \ldots] \\ &= r_t + \gamma\, \mathbb{E}(r_{t+1} \ldots + \gamma^{n-1} r_{t+n} + \ldots)] \\ &= r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) \end{aligned}$$

Q score of the current state depends recursively on the next state

Idea: recursively update Q score based on the estimation of the next state

$$Q(S_t, a_t) \leftarrow r + \gamma \max_a Q(S_{t+1}, a_{t+1})$$

More advanced update rules: Time-Difference (时序差分) Learning

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha\, [\, r + \gamma \max_a Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t)]$$

当前值　　观测值　　对未来的估计

Q-Learning算法

Build up a table of Q(s, a) scores as follows:

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

  Initialize s

  Repeat (for each step of episode):

    Choose a from s using policy derived from Q

    Take action a, observe r, s'

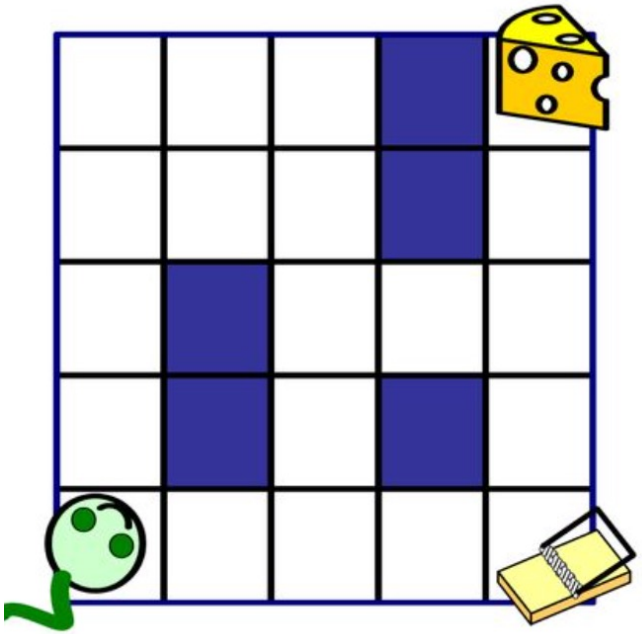
    Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

  $s \leftarrow s'$;

  Until s is terminal

Q-Learning举例：网格世界



MDP

- Agent
- States
- Transitions
- Reward

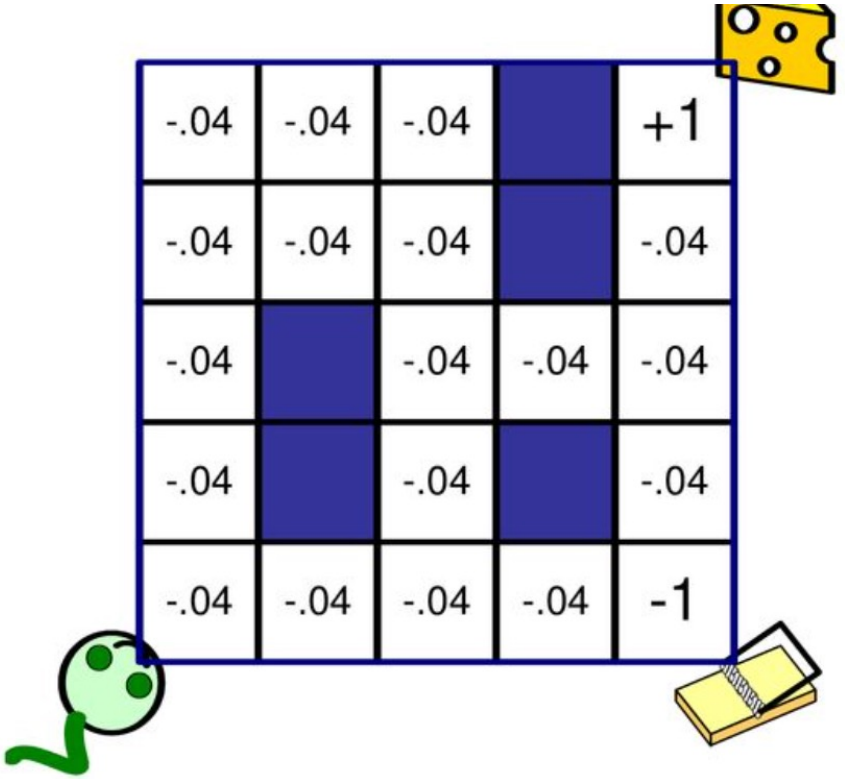
Use simple update rule (Q definition):

$$Q(S, a) \leftarrow r + \gamma \max_{a'} Q(S', a')$$

https://slideplayer.com/slide/16101998/

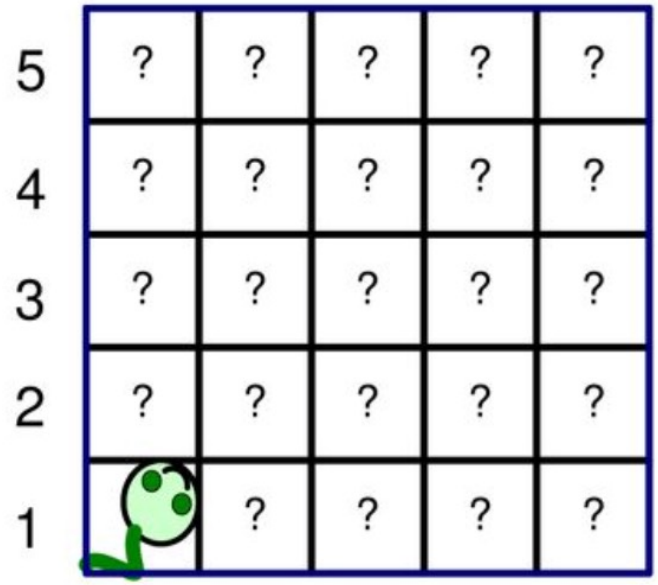
Q-Learning举例：网格世界

Reward r=-0.04 for every non-terminal state



Q-Learning举例：网格世界

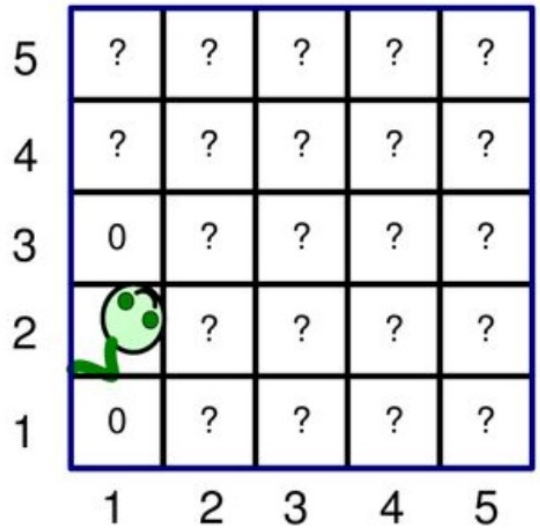
- Initialize Q-table and first position



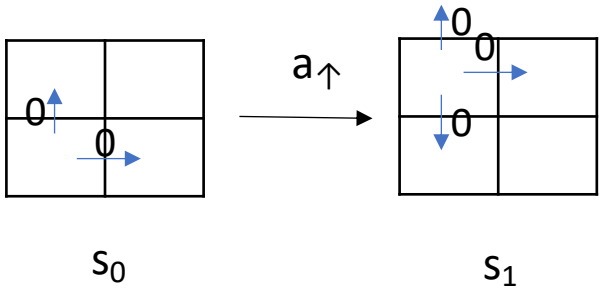
s	a	Q(s, a)
(1,1)	↑	0
(1,1)	→	0
(1,2)	↑	0
(1,2)	→	0
(1,2)	↓	0
…	…	0

Q-Learning举例：网格世界

- Move to the next step …

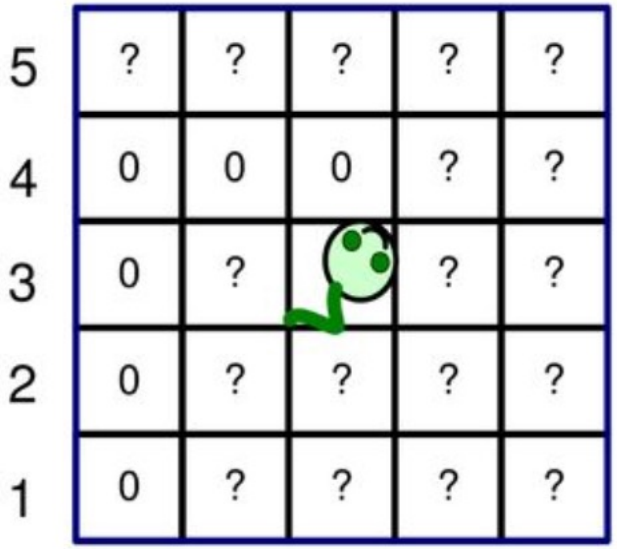


s	a	Q(s, a)
(1,1)	↑	-0.04
(1,1)	→	0.0
(1,2)	↑	-0.04



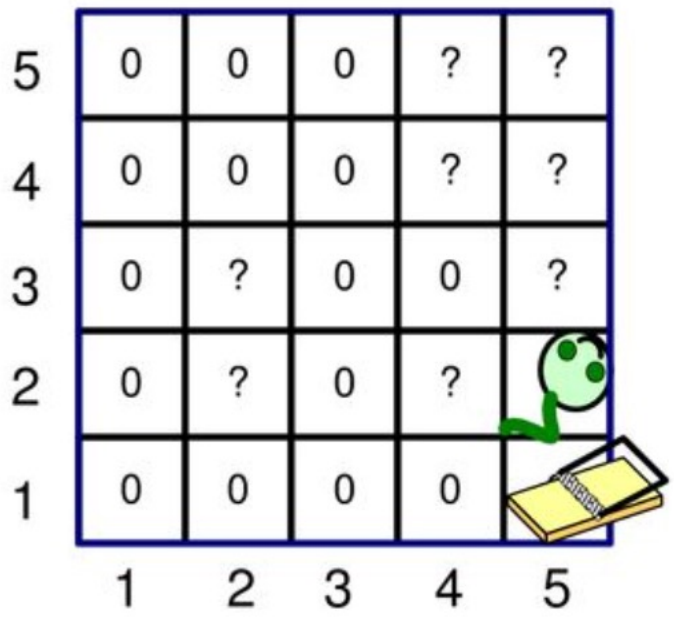
$Q(s_0, a_{\uparrow}) \leftarrow r + \gamma \max_{a'} Q(s_1, a')$
$\leftarrow -0.04 + 1 \max(0,0,0)$
$\leftarrow -0.04$

- Continue



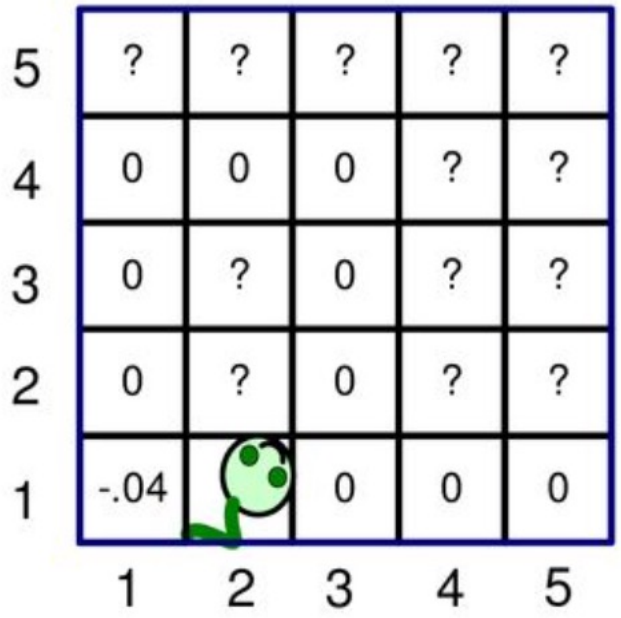
s	a	Q(s, a)
(1,1)	↑	-0.04
(1,1)	→	0.0
(1,2)	↑	-0.04
(1,3)	↑	-0.04
(1,4)	→	-0.04
(1,4)	↑	0.0
(2,4)	↓	-0.04

- Terminal state, start over



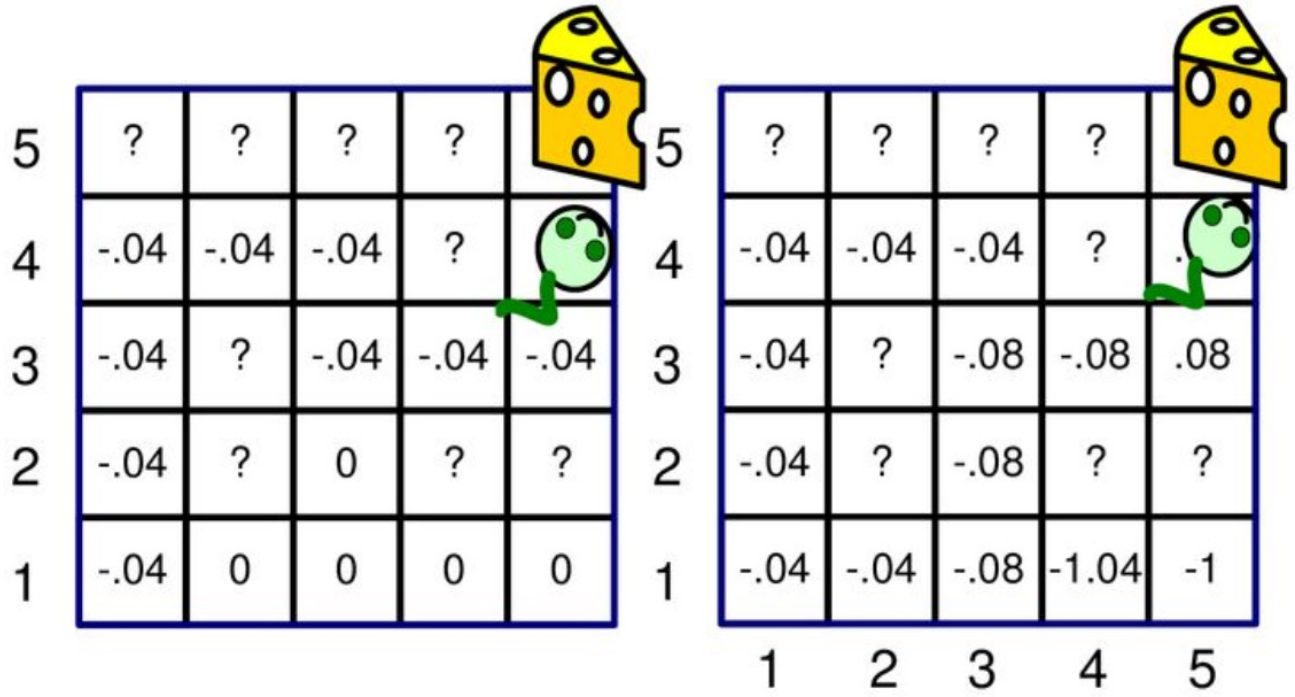
s	a	Q(s, a)
(1,1)	↑	-0.04
(1,1)	→	0.0
(1,2)	↑	
(1,4)	→	…
(1,3)	→	
(1,4)	→	-1.0
(1,5)	↑	-1.04

- Start a new iteration

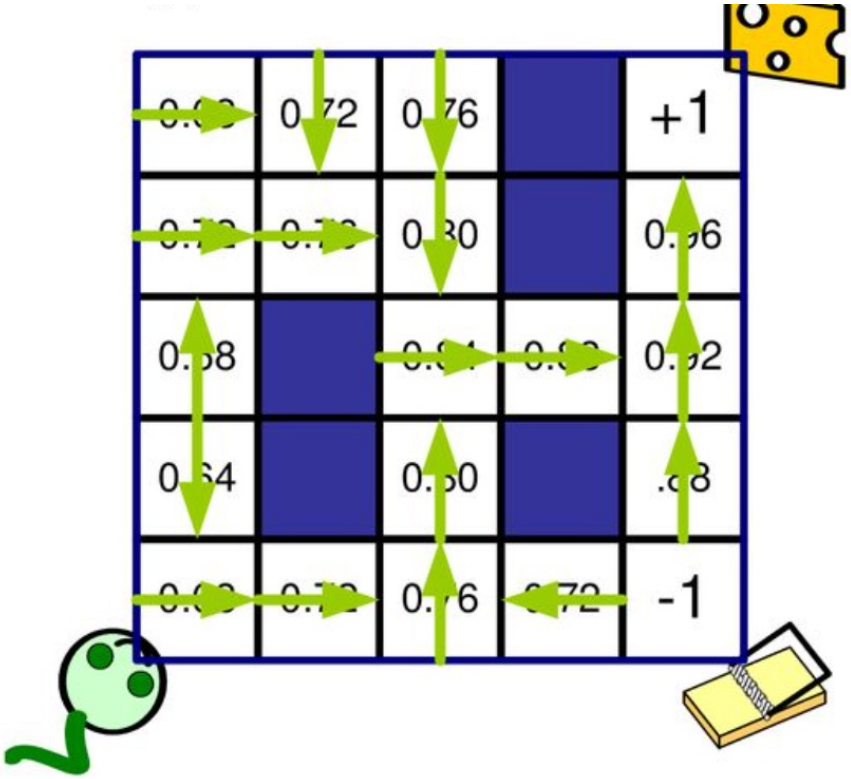


s	a	Q(s, a)
(1,1)	↑	-0.04
(1,1)	→	0.0+(-0.04)
(1,2)	↑	-0.04
(1,3)	↑	-0.04
(1,4)	→	-0.04
(1,4)	↑	0.0
…	…	…
(2,4)	↓	-0.04
(1,4)	→	-1.0
(1,5)	↑	-1.04

- After a few more iterations

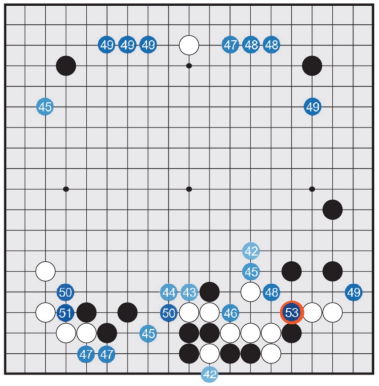


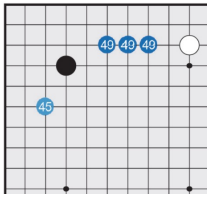
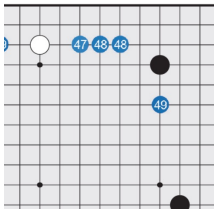
- The algorithm summarizes a policy of actions under each state (cell) based on the Q-table.



Q-Learning的局限：维度灾难

How about this scenario?

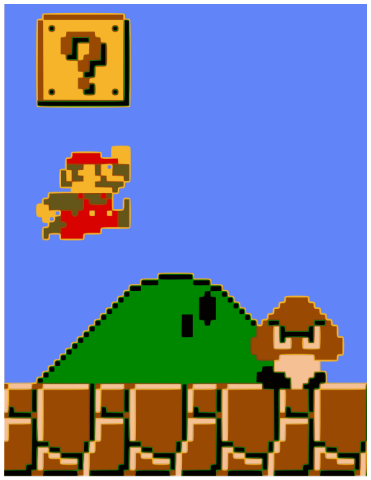


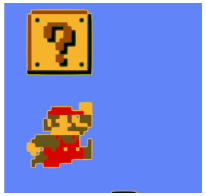
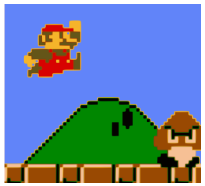
S	a	Q(S, a)
		14
		-10
⋮		

Infinite

Q-Learning的局限：维度灾难

And this



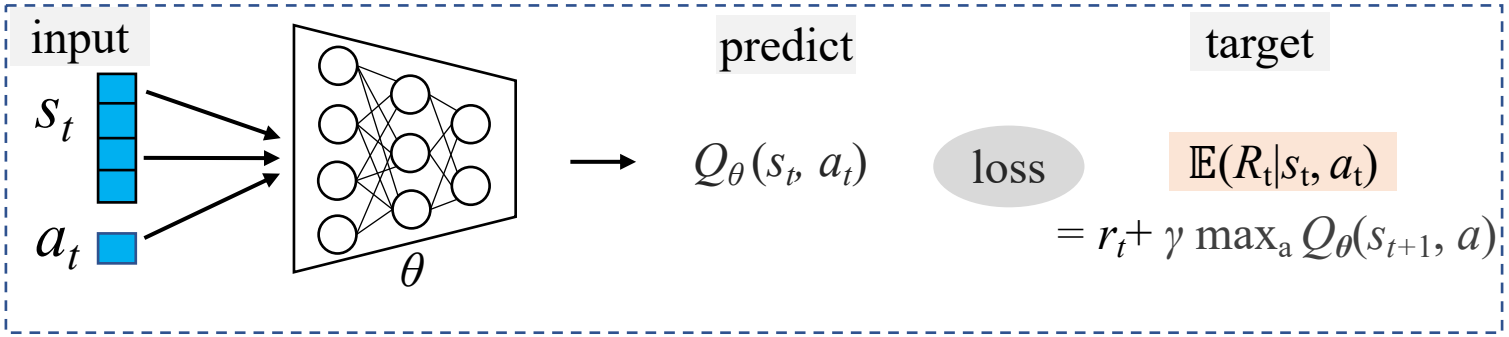
S	a	Q(S, a)
		12
		15
⋮		

Infinite

The states are usually intractable!

深度Q网络 (DQN)

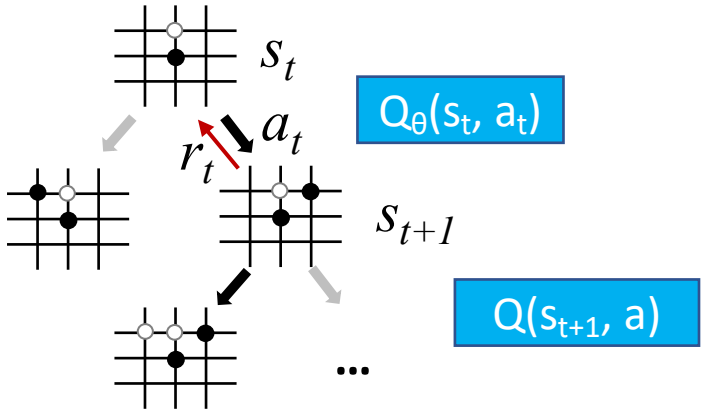
使用深度神经网络(如CNN)估计Q函数。



- **训练过程:**
  - 根据Agent的执行历史经验，估算Q网络的参数θ。
- **预测过程:**
  - 对于新观察的状态s, 计算所有动作的Q值 $Q_\theta(s, a)$ 。选择最优动作 $a = \text{argmax}_a Q_\theta(s, a)$.

数据和学习目标

- 局(**Episode**): 运行agent获得轨迹(s_1, a_1, r_1, s_2, a_2, r_2,…s_T, a_T, r_T)



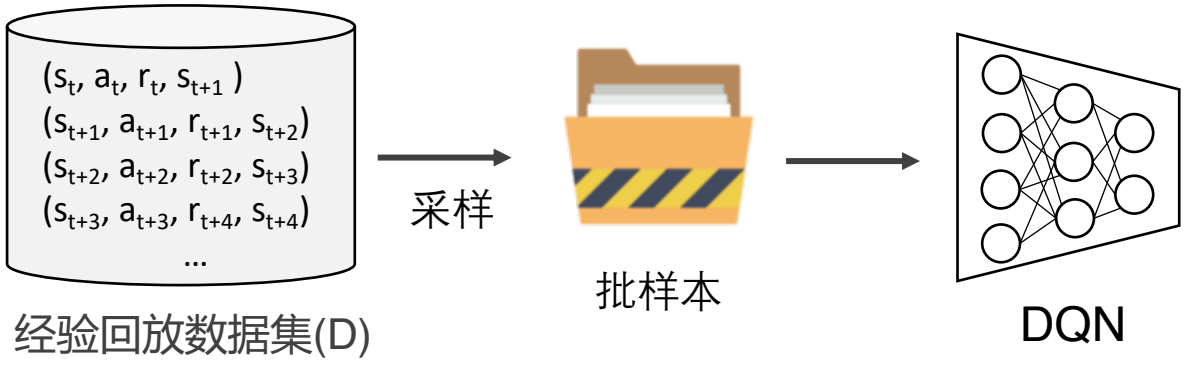
- 对任意输入s_t 和 a_t 模型预测一个Q分数$Q_\theta(s_t, a_t)$. 令 r_t 为该过程中Agent获得的奖励， s_{t+1} 是Agent进入的下一个状态。则该步骤Agent期望的最大长期奖励为: $r_t + \gamma \max_{a \in A} Q(s_{t+1}, a)$.
- 我们的目标是最小化预测值和目标值的差距, 也就是最小化均方损失函数(MSE):

$$\ell(\theta \mid s_t, a_t, r_t) = \| (r_t + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t) \|^2$$

target　　predicted

数据和学习目标

- **经验回放**: 将Agent执行多个回合, 将所有历史经验汇总为一个数据集: $D = \{(s^{(1)}, a^{(1)}, r^{(1)}, s'^{(1)}), \ldots, (s^{(N)}, a^{(N)}, r^{(N)}, s'^{(N)})\}$



- For multiple $(s, a, r, s') \in D$:

$$L(\theta \mid D) = \mathbb{E}_{(s,a,r,s')\sim D}[\| (r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a) \|^2]$$

mini-batch experience replay

Maximum possible Q-value for the next state (=Q_target)

Current predicted Q-value

训练方法

$$L(\boldsymbol{\theta} \mid D) = \mathrm{E}_{(\mathrm{s,a,r,s'})\sim \mathrm{D}}[\| (r + \gamma \max_{\mathrm{a'}} Q(s',a'; \theta) - Q(s, a; \theta)\|^2]$$

- 计算梯度:

$$\nabla_\theta L = \mathbb{E}_{(s,a,r,s')\sim D}\left(r + \gamma \max_{a'} Q(s', a'; \theta) - \boldsymbol{Q}(s, a; \theta)\right)\nabla_\theta \boldsymbol{Q}(s, a; \theta)$$

Q-target

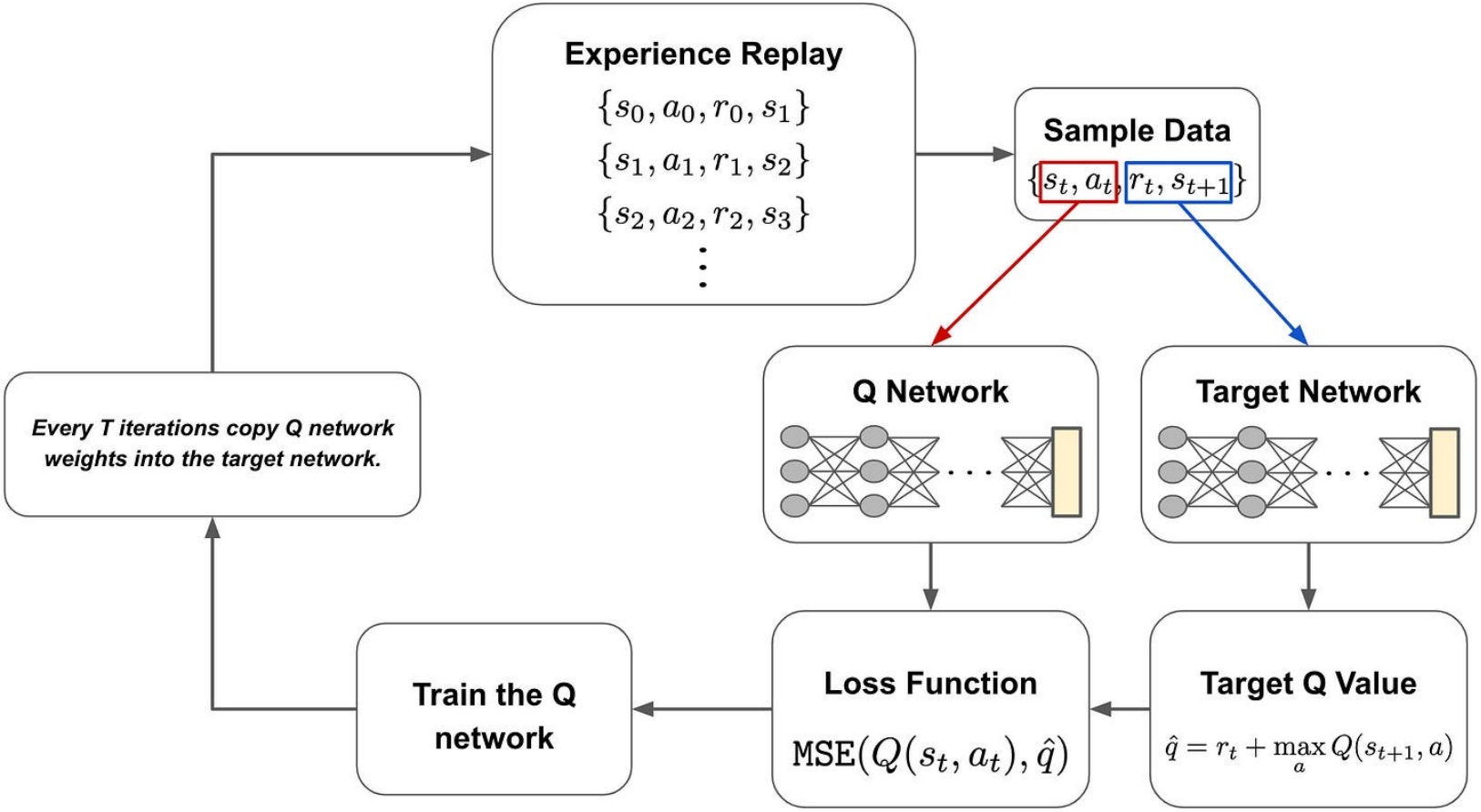
Problem: non-stationary target?

- **解决方案**: use some old, fixed parameters θ_{old} as a fixed Q-target (update every C steps)

$$\nabla_{\theta t} L = \mathbb{E}_{(s,a,r,s')\sim D}(r + \gamma \max_{a'} \boldsymbol{Q}(s', a'; \theta_{\text{old}}) - \boldsymbol{Q}(s, a; \theta t))\nabla_{\theta_t} \boldsymbol{Q}(s, a; \theta_t)$$

fixed, old Q-target

训练方法


Experience Replay
$\{s_0, a_0, r_0, s_1\}$
$\{s_1, a_1, r_1, s_2\}$
$\{s_2, a_2, r_2, s_3\}$
Sample Data
$\{s_t, a_t, r_t, s_{t+1}\}$
Q Network
Target Network
Every T iterations copy Q network weights into the target network.
Train the Q network
Loss Function
$\text{MSE}(Q(s_t, a_t), \hat{q})$
Target Q Value
$\hat{q} = r_t + \max_a Q(s_{t+1}, a)$

The Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

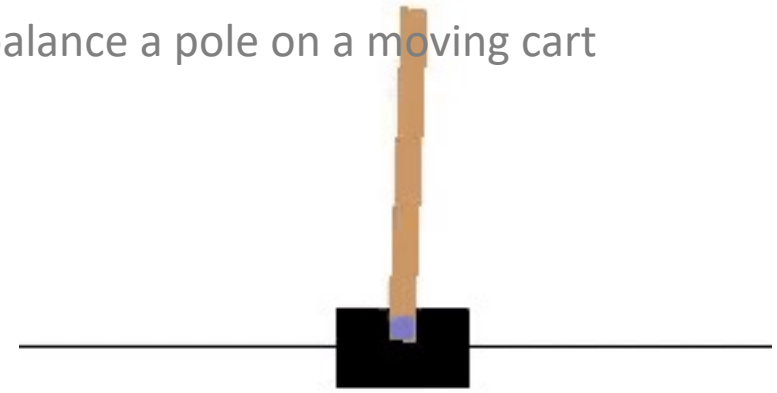
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for *episode* = 1,…, *M*
  Initialize state s_t
  for $t = 1, \ldots, T$
    With probability **ε** select a random action a_t — Sampling
    otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    $s_t \leftarrow s_{t+1}$
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from D
    Set $y_j = r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta)$ **if** s_{t+1} is non-terminal **else** r_j
    Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^2$ — Training
  end for
end for

Time for Coding

https://www.kaggle.com/code/dsxavier/dqn-openai-gym-cartpole-with-pytorch

CartPole

balance a pole on a moving cart



Reinforcement Learning Algorithms

Value Learning

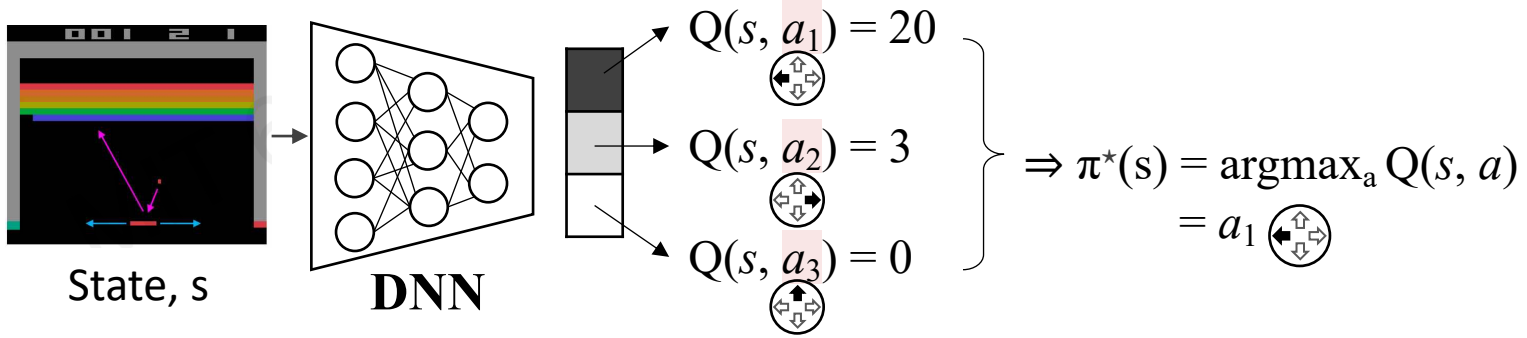
Find $Q(s, a)$
$a = \text{argmax}_a\, Q(s, a)$

Policy Learning

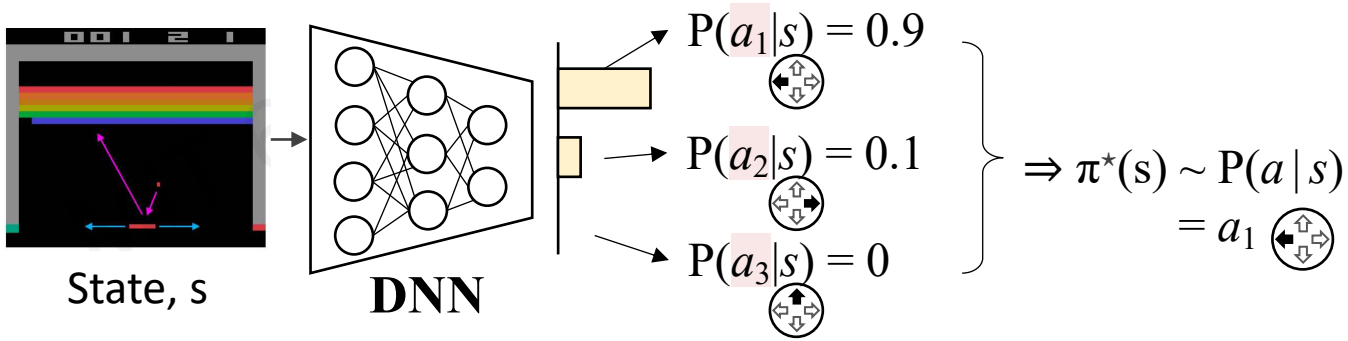
Find $\pi(s)$
Sample $a \sim \pi(s)$

Policy Gradient: Key Idea

DQN: Approximate a Q-function and use it to infer the optimal policy $\pi^*(s)$

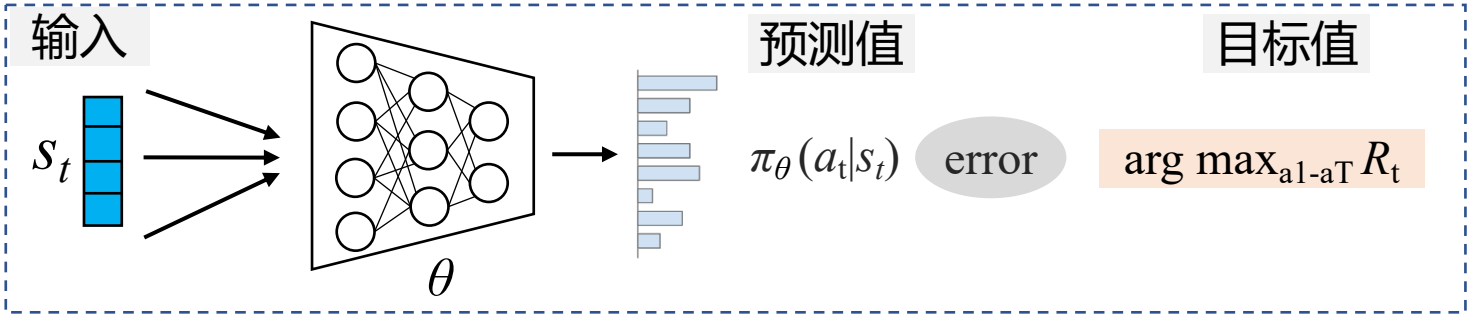


Policy Gradient: directly optimize the policy $\pi(s)$



策略网络

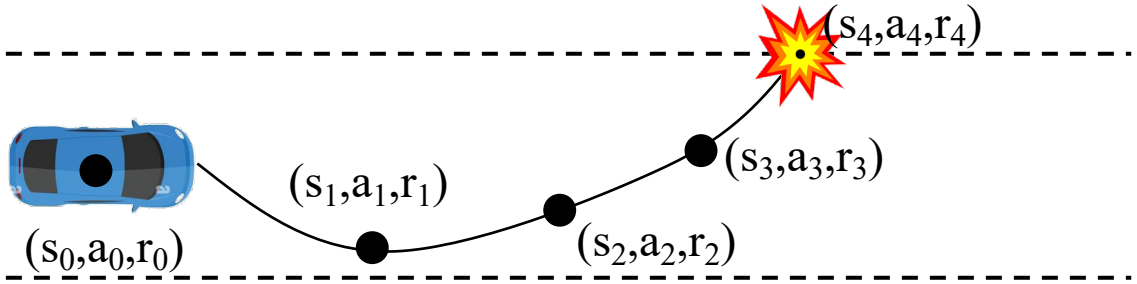
- 采用深度神经网络预测策略 π_θ.



- Train:
  - Estimate the parameters $\boldsymbol{\theta}$ in the $\pi_\theta(a|s)$ network using agent play experiences.
- Test:
  - choose $a \sim \pi_\theta(\mathrm{a} \mid s)$.

数据和学习目标

- **Episode**: run a policy π_θ until termination, record the trajectory $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T)$.



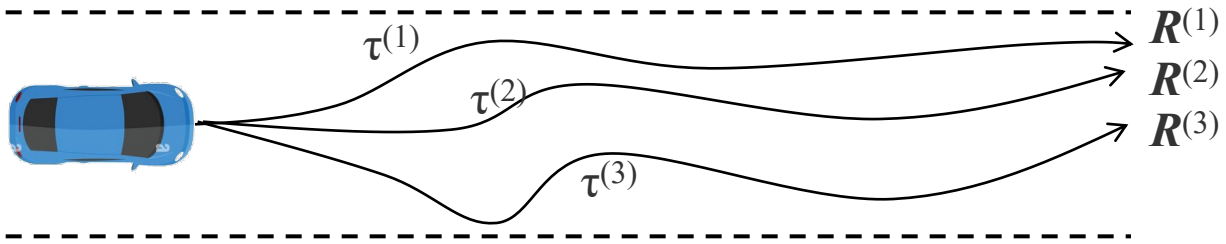
- The reward for τ is:

$$\boldsymbol{R}(\tau) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \gamma^{T-t} r_T$$

τ Is stochastic, therefore R(τ) is stochastic

数据和学习目标

- 按照策略 π_θ,运行Agent若干回合，获得多个轨迹 $\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(N)} \sim \pi_\theta(a|s)$

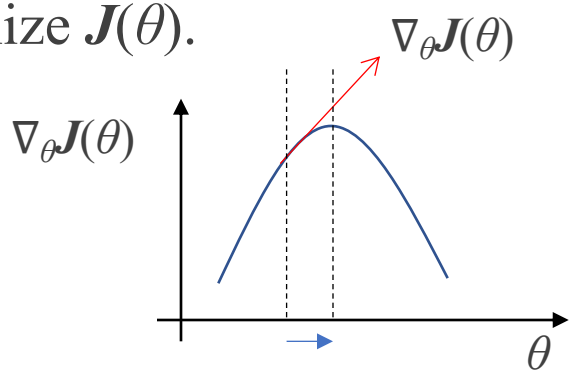


- The expected accumulated reward following the policy π_θ

$$J(\theta) = \mathbb{E}_{\tau\sim p(\tau)} R(\tau) = \Sigma_\tau R(\tau) p_\theta(\tau)$$

- Our goal is to find θ that maximize $J(\theta)$.

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$



数据和学习目标

推导目标函数的梯度

$$\nabla_\theta J(\theta) = \Sigma_\tau R(\tau)\nabla p_\theta(\tau) = \sum_\tau R(\tau)\, p_\theta(\tau)\frac{\nabla p_\theta(\tau)}{p_\theta(\tau)}$$

$$= \sum_\tau R(\tau)\, p_\theta(\tau)\nabla \log p_\theta(\tau) \;= \mathbb{E}_{\tau\sim p(\tau)}[R(\tau)\nabla \log p_\theta(\tau)\,]$$

$$\approx \frac{1}{N}\sum_{l=1}^{N} R^{(l)}\nabla \log p_\theta(\tau^{(l)}) = \frac{1}{N}\sum_{l=1}^{N}\sum_{t=1}^{T} R^{(l)}\nabla \log \pi_\theta(a_t^{(l)}\,|\,s_t^{(l)})$$

τ is a stochastic process with a probability:

$p_\theta(\tau)=p(s_1)\pi_\theta(a_1|s_1)p(s_2|s_1, a_1)\pi_\theta(a_2|s_2)p(s_3|s_2,a_2)\ldots$

Trainable

deterministic and assigned by the environment

The Algorithm

代码片段17.3 REINFORCE算法 （Williams 1992）

采样轨迹： 根据当前策略$\pi_\theta(a|s)$让智能体在环境中运行若干局，采样完整的轨迹$\{\tau^{(1)},\ldots,\tau^{(N)}\}$。

计算策略梯度： 对于第i条采样到的轨迹$\tau^{(i)}$:

1. 计算轨迹中每个状态-动作对 $(s_t^{(i)}, a_t^{(i)})$ 的对数概率

$$\log \pi_\theta\left(a_t^{(i)} \middle| s_t^{(i)}\right)$$

2. 对上一步的对数概率结果求关于参数θ的梯度

$$\nabla_\theta \log \pi_\theta\left(a_t^{(i)} \middle| s_t^{(i)}\right)$$

3. 将计算出的梯度乘以一个权重，该权重为这条轨迹的总回报 $R(\tau^{(i)})$。

梯度更新： 使用计算出的平均策略梯度更新参数：

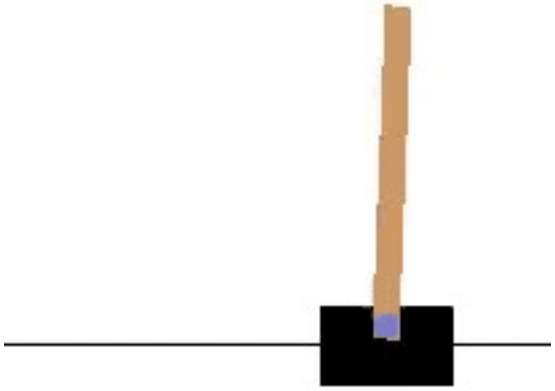
$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

重复步骤1至3，直到策略收敛

TIME for Coding

Tutorial: CartPole

balance a pole on a moving cart



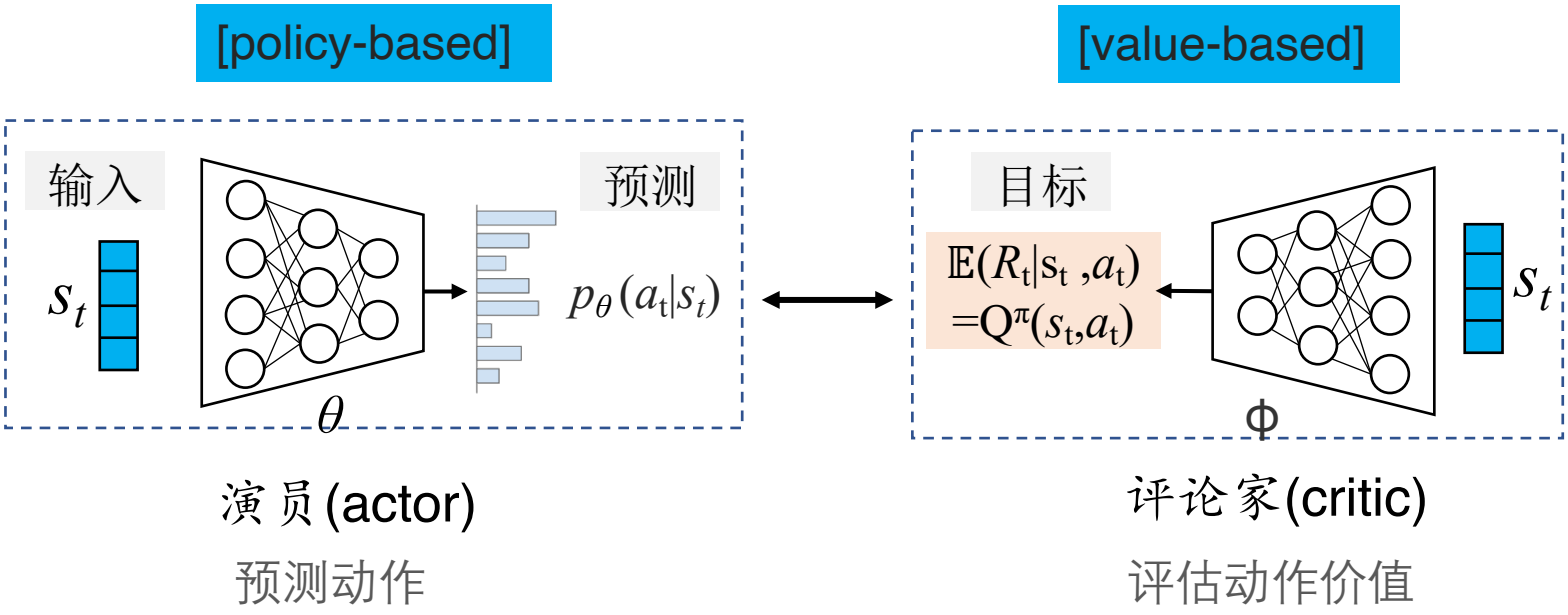
https://www.kaggle.com/code/abedsultan/cartpole-balance-with-reinforce-policy-gradient-rl

演员评论家算法Actor-Critic

- **融合策略学习和值学习**

  Actor: predicts the action at under each state s_t

  Critic: estimates the long-term return $\mathbb{E}(\boldsymbol{R}_t \mid s_t, a_t)$ for the actor



训练Actor-Critic

损失(目标)函数:

Policy Gradient: $J_{act}(\theta) = \Sigma_t \mathbf{R} \log \pi_\theta(a_t|s_t)$

Actor-critic: $J_{act}(\theta) = \Sigma_t \mathbf{Q}(s_t, a_t) \log \pi_\theta(a_t|s_t)$ Q Actor-Critic

$Q(s_t, a_t)$ have large values
Let's subtract its average, $V(s_t)$

$J_{act}(\theta) = \Sigma_t [\mathbf{Q}(s_t, a_t) - \mathbf{V}(s_t)] \log \pi_\theta(a_t|s_t)$ TD Actor-Critic

优势函数

Advantage Function (优势函数)

引入优势函数，表示选择的动作相对平均动作的优势：

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

- $A^{\pi}(s,a) > 0$，说明动作a比平均水平好，是个“惊喜”，值得鼓励(增加该动作的概率)。
- $A^{\pi}(s,a) < 0$，说明动作a比平均水平差，应该避免（降低该动作的概率）。

新的目标函数：

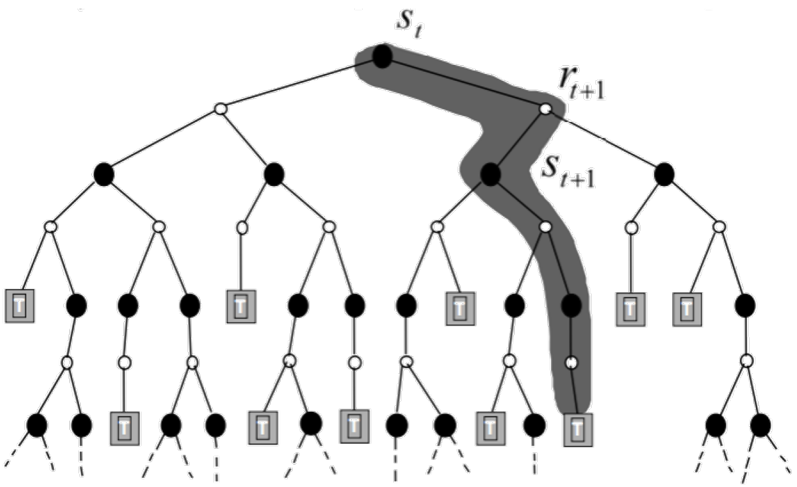
$$\nabla_{\theta} J(\theta) = E_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) A^{\pi}(s_t, a_t) \right]$$

状态值和TD误差

- 状态-值函数$V^{\pi}(s)$ 刻画了Agent在观察到状态s后的期望累计奖励。

$$V^{\pi}(s) = \mathbb{E}(R_t|s)$$

score(state)



$V^{\pi}(s)$ 较大

$V^{\pi}(s)$ 较小

Temporal-difference error

$$V^{\pi}(s_t) \approx r + \gamma V^{\pi}(s_{t+1})$$

smaller variance　may be biased

$$\delta = r + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

用时序差分误差替代优势函数

Actor: 依然是策略梯度，将期望回报换成TD误差

$$J(\theta) = \Sigma_t \, [\boldsymbol{Q}(s_t, a_t) - \mathbf{V}(s_t)] \log \pi_\theta(a_t|s_t)$$

$$= \Sigma_t \underbrace{[\mathbf{r} + \gamma \mathbf{V}(s_{t+1}) - \mathbf{V}(s_t)]}_{\text{TD误差 } \delta} \log \pi_\theta(a_t|s_t)$$

$$\nabla J(\theta) = \Sigma_t \, \delta_t \, \nabla \log \pi_\theta(a_t|s_t)$$

时序差分误差（TD Error, δ_t），恰好是优势函数$A^\pi(s,a)$的一个无偏估计！

Critic: 类似Q-learning,作值回归

$$L(\varphi \mid D) = \mathbb{E}_{(s,r,s')\sim D}[\| (r + \gamma V(s'; \varphi) - V(s; \varphi)\|^2]$$

$$\nabla_{\varphi t} L = \mathbb{E}_{(s,r,s')\sim D}(\underbrace{r + \gamma V(s'; \varphi_{\text{old}})}_{\textbf{fixed, old V-target}} - V(s; \varphi_t)) \nabla_{\varphi_t} V(s; \varphi_t)$$

The Algorithm

基于优势函数的Actor-Critic算法（A2C)

1. 初始化actor网络$\pi_\theta(a|s)$和critic网络$V_\varphi(s)$
2. for episode =1 to M do:
3. 从环境获得初始状态s_0
4. for t=0 to T do：
5. Actor根据当前策略采样并执行动作$a_t \sim \pi_\theta(a|s_t)$
6. 环境返回即时奖励r和下一时刻状态s_{t+1}
7. 计算TD目标(Target): $y_t = r + \gamma V_\phi(s_{t+1})$
8. 计算TD误差(Advantage): $\delta_t = y_t - V_\phi(s_t)$

更新critic参数φ (最小化均方误差): $\min_\varphi \delta_t^2$

更新actor参数θ (梯度上升)： $\theta \leftarrow \theta + \eta \nabla_\theta \log[\pi_\theta(a_t|s_t)] \cdot \delta t$

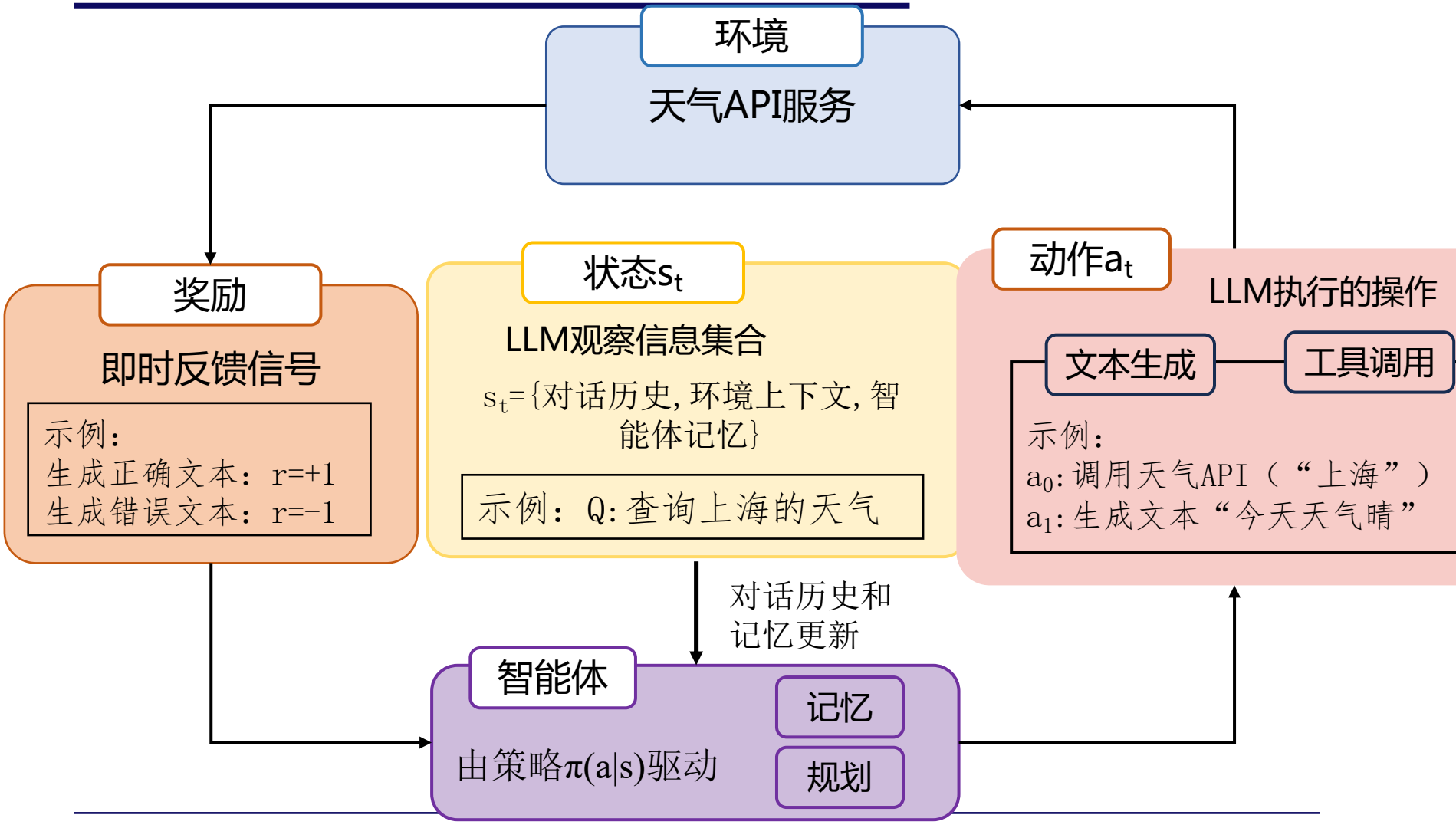
$s_t \leftarrow s_{t+1}$

end for

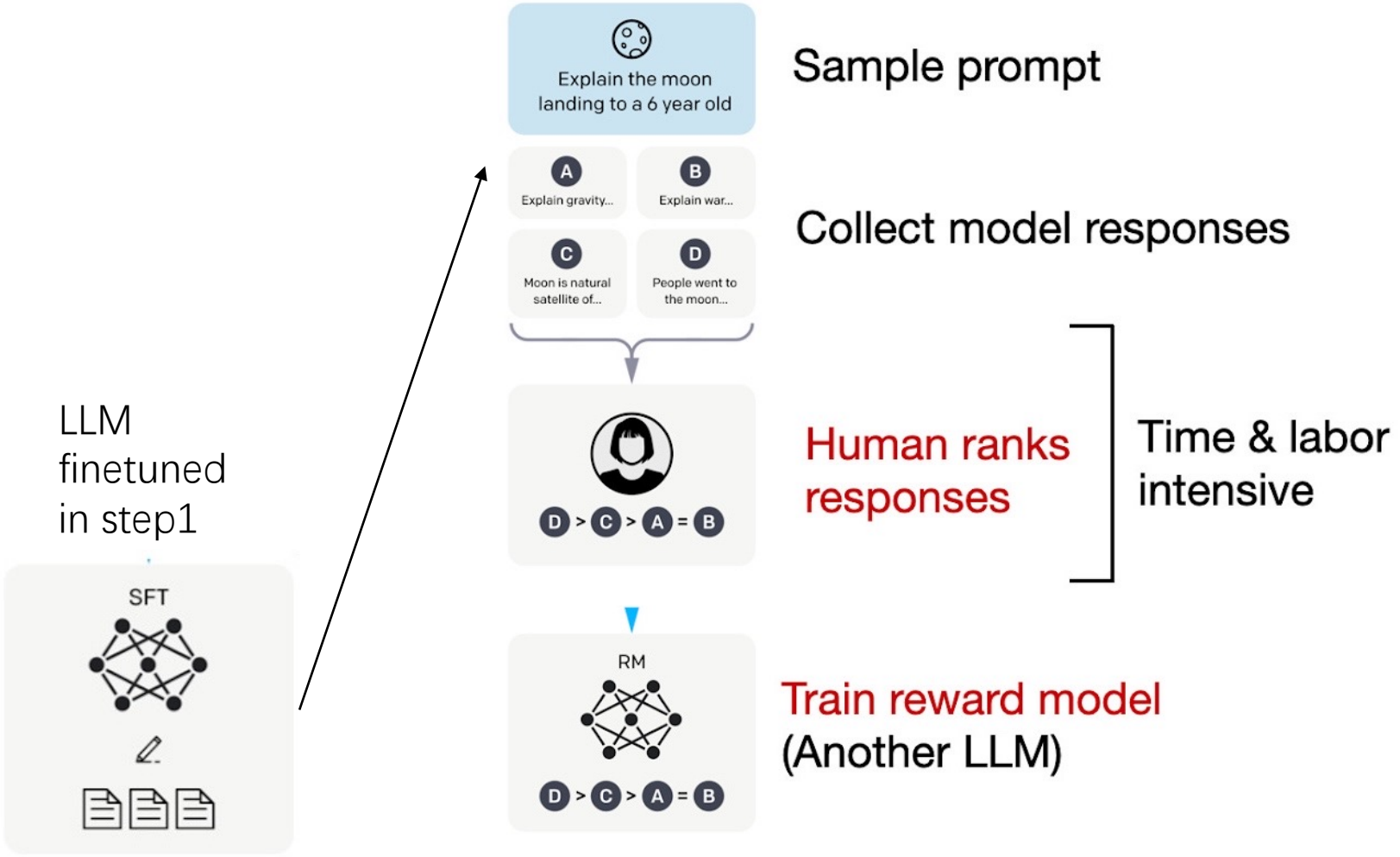
end for

强化学习与大语言模型

LLM Agent的MDP设定

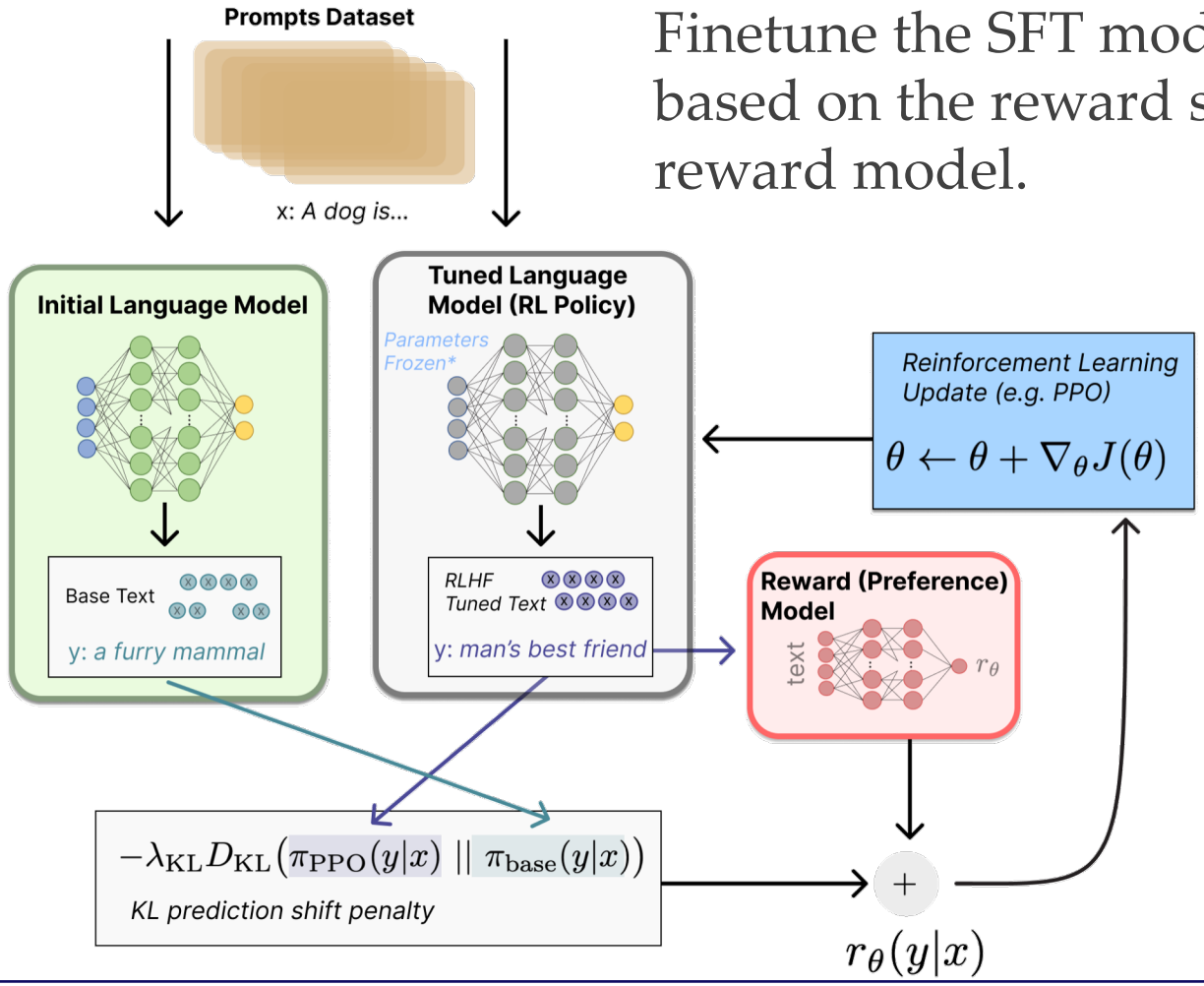


Phase1: Create a Reward Model



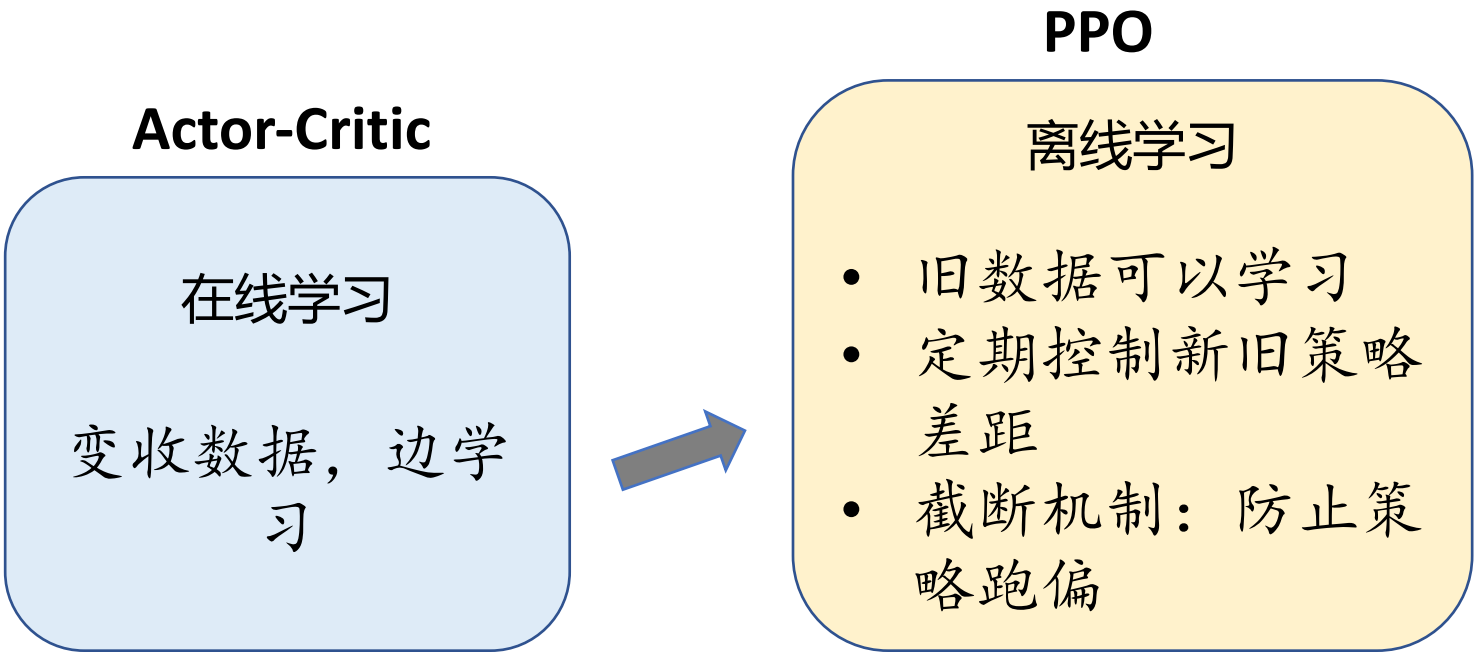
Phase2: Finetuning via RL

Finetune the SFT model using RL based on the reward scores from the reward model.



Proximal Policy Optimization (PPO)

Proposed by OpenAI in 2017



Proximal Policy Optimization (PPO)

1. 离线学习策略：

引入旧策略$\pi_{\theta old}(a|s)$, 模型根据旧策略采集轨迹数据

2. 新旧策略对齐：

引入策略比率，

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

r_t>1:相对于旧策略，新策略更倾向于在状态s_t下采取动作a_t。

引入代理目标函数：

$$L^{CPI}(\theta) = \mathbb{E}_t[r_t(\theta)A_t]$$

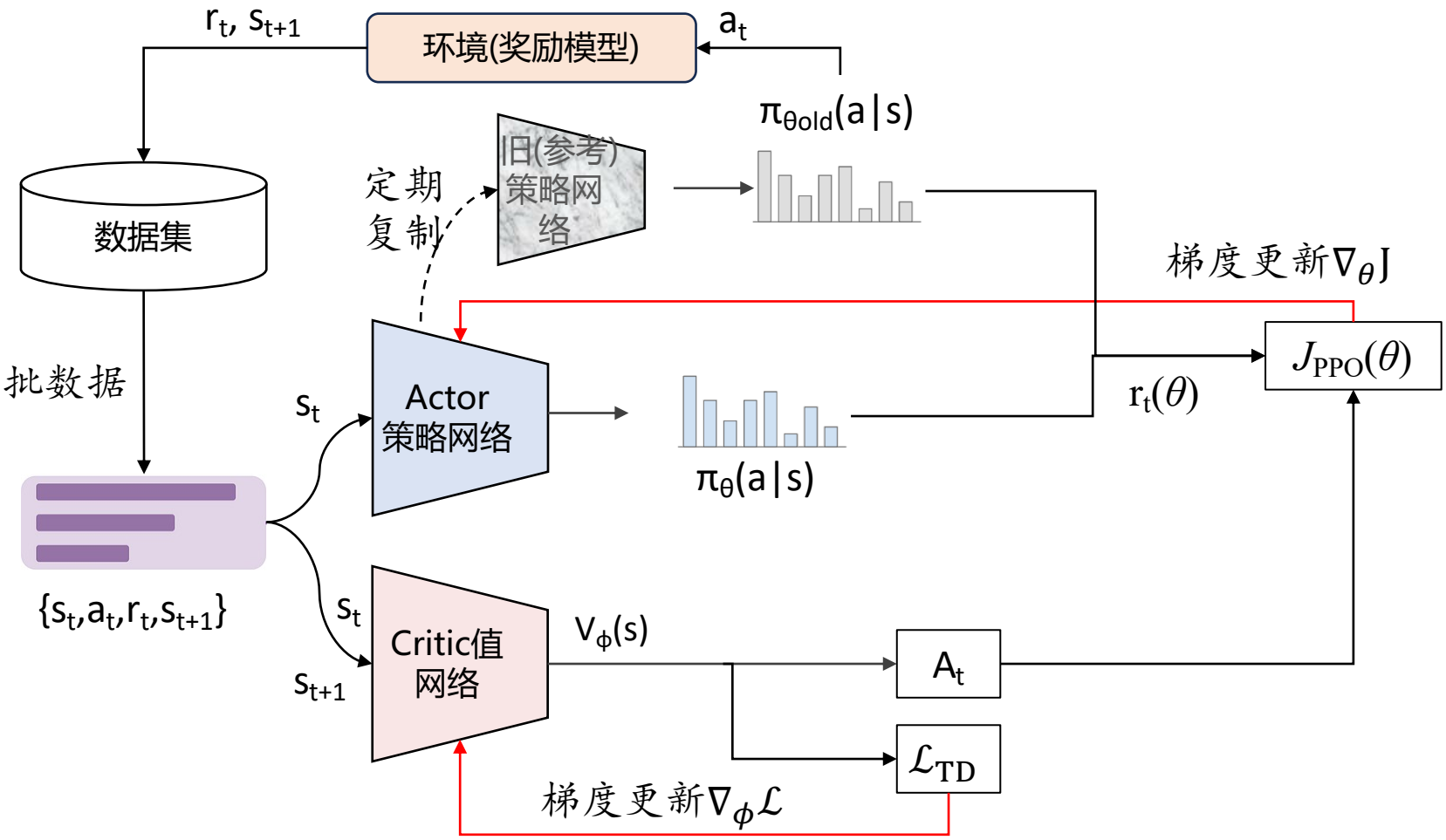
如果动作是好的（$A_t > 0$），最大化L会促使比率r_t变大（新策略更倾向于该动作）

3. 截断机制：

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \mathbf{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

$\text{clip}(x, \min, \max)$是一个剪裁操作，强行将变量$x$限制在$[1-\epsilon, 1+\epsilon]$

Proximal Policy Optimization (PPO)



PPO算法流程

初始化$\pi_\theta(a|s)$和critic网络$V_\varphi(s)$

for i =1 to K do:

// 数据收集阶段

保存当前的Actor策略参数：$\theta_{old} \leftarrow \theta$

使用旧策略$\pi_{\theta_{old}}$在环境中运行T个时间步，收集轨迹数据$\{s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T\}$。

使用Critic网络$V_\varphi(s)$计算这批数据的目标价值和优势函数估算值A_t。

// 网络更新阶段

for epoch = 1 to N do:

从收集的数据中随机打乱，划分为若干个小批量 (Mini-batch)。

对每个批数据$B=\{s_t,a_t,r_t,s_{t+1}\}$执行：

计算策略比率：$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$

计算Actor的截断目标函数：

$$L^{Actor}(\theta) = \frac{1}{|B|}\sum \min(r_t A_t, clip(r_t, 1-\epsilon, 1+\epsilon)A_t)$$

计算Critic的价值损失 (均方误差)：

$$L^{Critic}(\phi) = \frac{1}{|B|}\sum \left(r + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)\right)^2$$

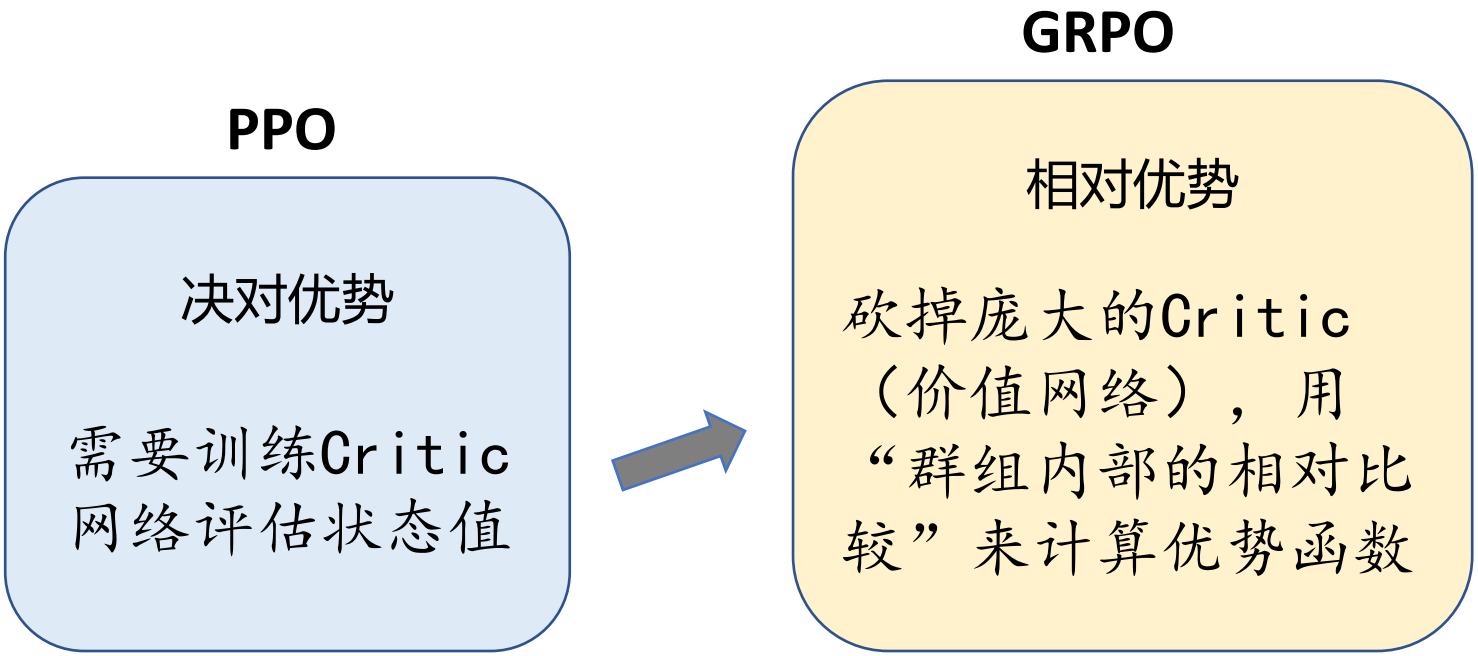
通过梯度上升最大化$L^{Actor}(\theta)$更新θ；通过梯度下降最小化 $L^{Critic}(\phi)$ 更新φ。

end for

end for

GRPO

群组相对策略优化(Group Relative Policy Optimization)



GRPO

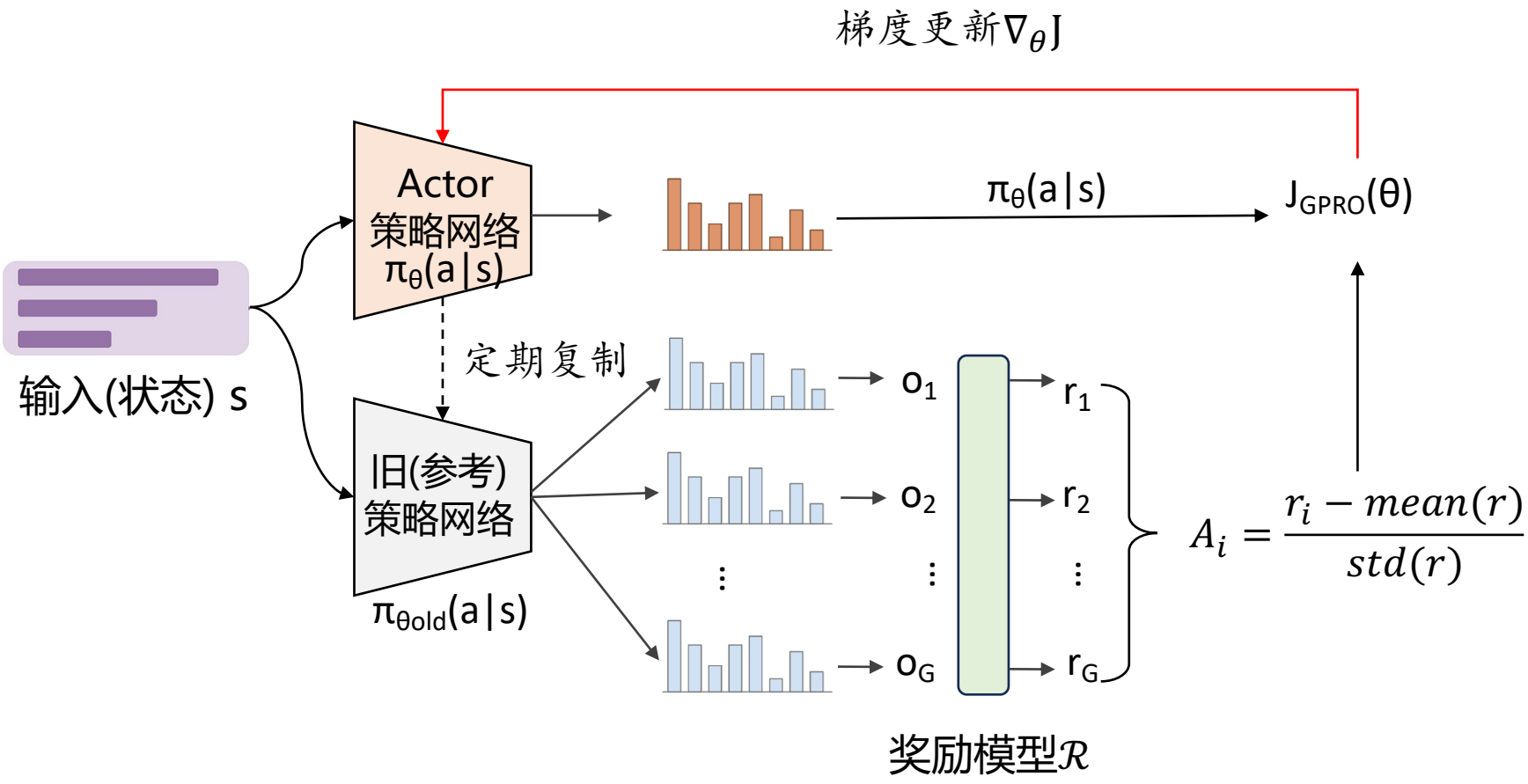
1. 策略网络生成多个候选结果（群组）{o_1, o_2, ..., o_G}，每个对应奖励{r_1,r_2,...,r_G}
2. 计算每个结果在群组中的相对优势

$$A_i = \frac{r_i - \mathrm{mean}(r_{1:G})}{std(r_{1:G})}$$

3. 与参考网络(通常是SFT后的大模型)参数对齐

$$J_{\mathrm{GRPO}}(\theta) = \mathbb{E}[\frac{1}{G}\sum_{i=1}^{G}(min(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, clip(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\epsilon, 1+\epsilon)A_i) - \beta\mathbb{D}_{KL}(\pi_\theta||\pi_{ref}))]$$

GRPO


梯度更新$\nabla_\theta J$
Actor
策略网络
$\pi_\theta(a|s)$
$\pi_\theta(a|s)$
$J_{GPRO}(\theta)$
输入(状态) s
定期复制
旧(参考)
策略网络
$\pi_{\theta old}(a|s)$
o_1
o_2
o_G
r_1
r_2
r_G
$A_i = \frac{r_i - mean(r)}{std(r)}$
奖励模型$\mathcal{R}$