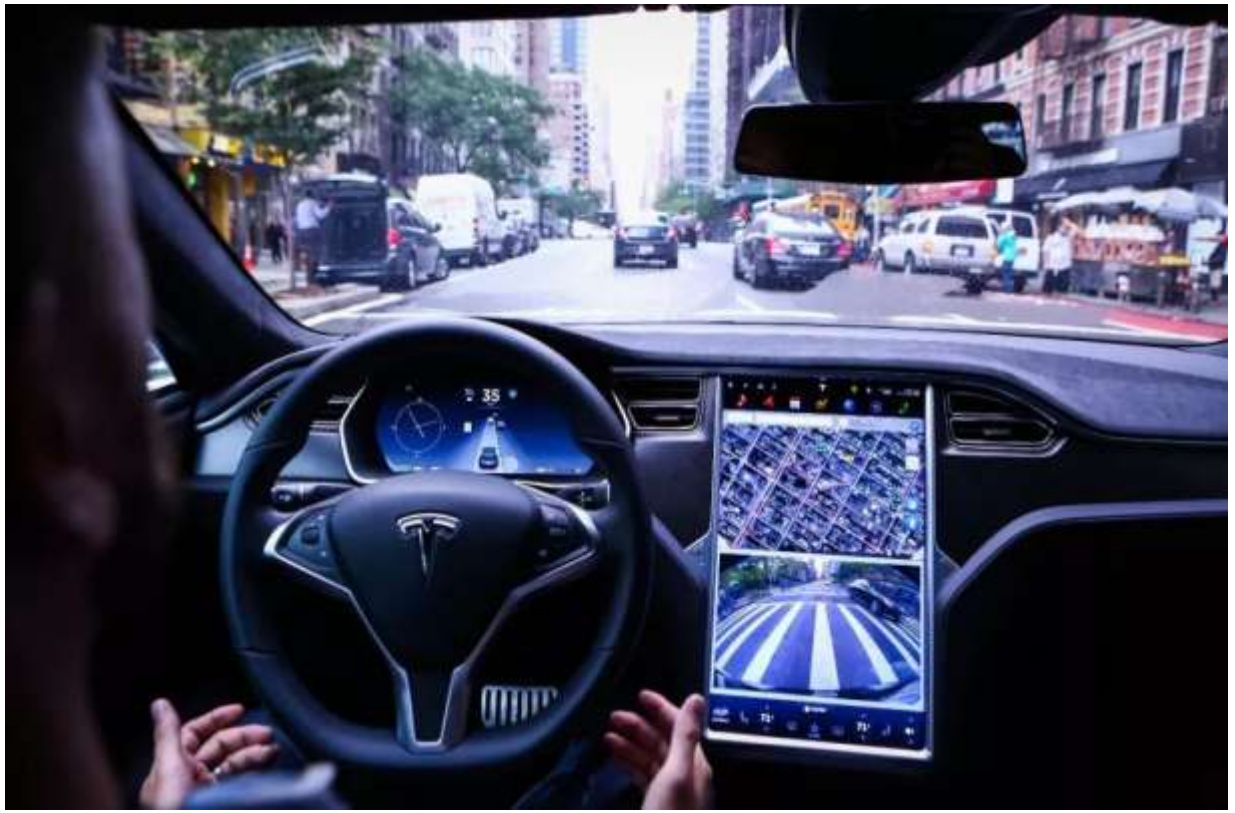
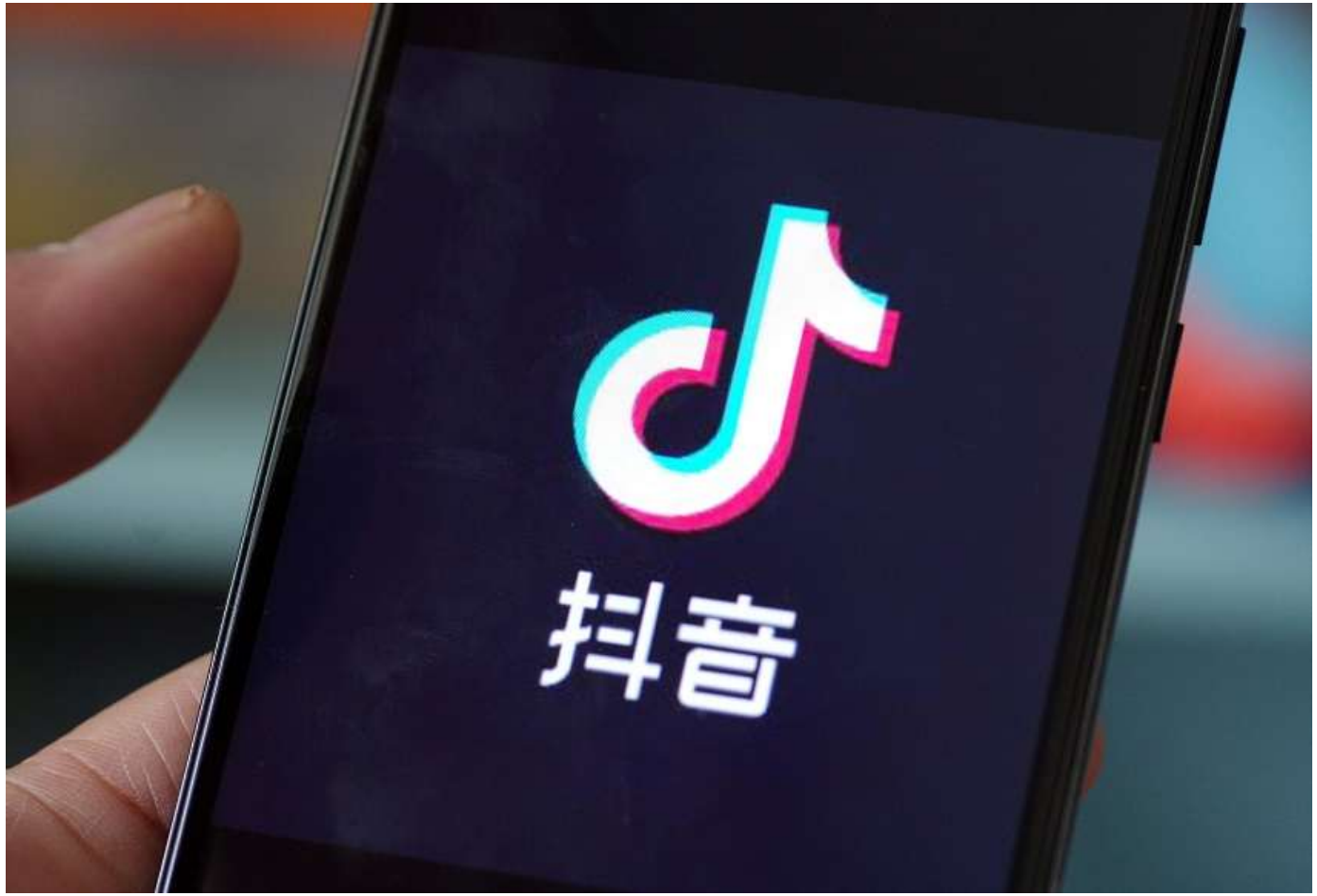
第16章 强化学习

汇报人：陈程
时间：2025.04.13

阿尔法围棋（AlphaGo）



特斯拉自动驾驶



每次滑动，AI都在实验：

- 推新内容=探索（可能会失去用户）
- 推老内容=利用（可能陷入信息茧房）

这就是强化学习的**探索-利用困境（Exploration-Exploitation Tradeoff）**。

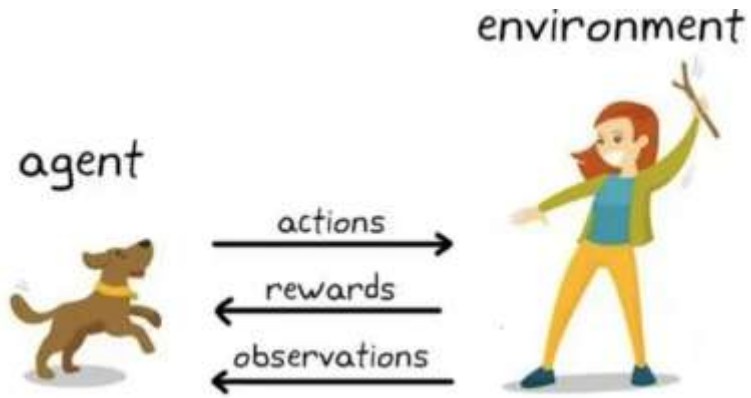
目录

概念

强化学习（Reinforcement Learning，RL）是机器学习的一个分支，其核心思想是让智能体（Agent）在与环境（Environment）的交互中学习，通过试错的方式，找到能够获得最大累积奖励（Cumulative Reward）的最优策略（Policy）。

基本元素

概念	定义
智能体（Agent）	强化学习中的决策者，通过执行动作与环境进行交互。
环境（Environment）	智能体外部的世界，响应智能体的动作，并提供新的状态和奖励。
状态（State）	对环境当前情况的描述，智能体根据状态决定下一步的动作。
动作（Action）	智能体在给定状态下可以执行的操作。
奖励（Reward）	环境对智能体动作的反馈，衡量动作的好坏。智能体的目标是最大化累积奖励。
策略（Policy）	智能体在给定状态下选择动作的规则，可以是确定性的或随机性的。
价值函数（Value Function）	用于评估给定状态下或执行给定动作的长期价值。包括状态价值函数和动作价值函数。



简单来说，强化学习就是要建立一种**奖励机制**，然后不断试错，每一次试错后，让自己更靠近到那些返回更多奖励的尝试点。

区别

维度	监督学习 (Supervised)	无监督学习 (Unsupervised)	强化学习 (Reinforcement)
数据形式	带标签的 (输入→输出) 样本	无标签的原始数据	状态、动作、奖励序列
目标	最小化预测误差	发现数据中的隐藏结构	最大化长期累积奖励
是否需要环境交互	❌ 静态数据集	❌ 静态数据集	✅ 动态环境交互
反馈类型	即时、明确的标签	无明确反馈	延迟、稀疏的奖励信号
典型任务	图像分类、语音识别	聚类、降维	游戏AI、机器人控制、自动驾驶决策

马尔可夫决策过程

马尔可夫决策过程（Markov decision processes,MDP)

- MDP提供了一套为在结果部分随机、部分在决策者的控制下的决策过程建模的数学框架。

- MDP的核心特征

  - 完全可观测性：智能体感知的观察值=环境真实状态（$O_t = S_t$）
  - 状态的充分统计性（马尔可夫性质）：当前状态S_t是预测未来的充分统计量，历史信息可丢弃。

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

历史信息（S_1到S_{t-1}）对预测未来毫无帮助

这极大地简化了问题

马尔可夫决策过程（Markov decision processes,MDP)

MDP可以是由一个五元组表示（S,A,{P_{sa}}, γ ,R)

- S是状态的集合
- A是动作的集合
- P_{sa}是状态转移概念

  对每个状态s∈S和动作a∈A，P_{sa}是下一个状态在S中的状态分布
- $\gamma \in$ [0,1]是对未来奖励的折扣因子
- R：S × A→ R是奖励函数

马尔可夫决策过程（Markov decision processes,MDP)

MDP的动态如下所示：

- 从状态s_0开始
- 智能体选择某个动作$a_0 \in A$
- 智能体得到奖励$R(s_0,a_0)$
- MDP随机转移到下一个状态$s_1 \sim P_{s0a0}$
- 这个状态不断进行

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

- 直到终止状态s_T出现为止，或者无止尽地进行下去
- 智能体的总回报为

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

基于动态规划的强化学习

价值函数与贝尔曼方程

状态价值函数$V^{\pi}(s)$

定义：

在策略 π 下，从状态 s 出发能获得的长期回报期望值。

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s\right]$$

未来所有奖励的折现和　当前状态=s

其中，

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- $R_{t+1}, R_{t+2}, \ldots$：从时刻 t 开始获得的即时奖励序列。
- $\gamma \in [0, 1]$：折扣因子，决定未来奖励的权重。

状态价值函数的贝尔曼方程：

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^{\pi}(s')\right]$$

策略加权　状态转移概率　即时奖励　未来折扣价值

回报

价值函数与贝尔曼方程

动作价值函数$Q^{\pi}(s,a)$

定义：

在状态 s 下执行动作 a，之后遵循策略 π 所能获得的期望总回报（包括即时奖励和未来折扣奖励）。

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s, A_t = a\right]$$

动作状态函数的贝尔曼方程：

$$Q^{\pi}(s,a) = \sum_{s'} P(s' \mid s,a)\left[R(s,a,s') + \gamma \sum_{a'} \pi(a' \mid s') Q^{\pi}(s',a')\right]$$

执行动作 a 后转移到状态s’的概率

即时奖励

未来奖励的期望值

价值函数与贝尔曼方程

最优状态价值函数V*(s):

从状态 s 出发，所有可能策略中能获得的最大长期回报。

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

最优动作价值函数Q*(s,a):

在状态 s 下执行动作 a，之后遵循最优策略，能获得的最大长期回报。

$$V^*(s) = \max_{a} Q^*(s,a)$$

贝尔曼最优方程：

$$V^*(s) = \max_{a} \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

V*(s)方程：当前状态的最优价值 = 所有动作中，选即时奖励 + 未来最优价值折现的最大值

Q*(s)方程：当前动作的最优价值 = 即时奖励 + 所有可能下一状态的最优动作价值的折现期望

策略迭代

1.策略评估（Policy Evaluation）
（假设我们有一个初始策略 π（比如随机策略），如何知道它好不好？）

方法：用贝尔曼方程迭代计算V^{π}

$$V_{k+1}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V_k(s') \right]$$

反复用当前策略模拟环境，更新各状态的价值，直到收敛

2.策略改进（Policy Improvement）
（有了V^{π}，如何让策略变得更优？）

方法：对每个状态s，选择能使$Q^{\pi}(s,a)$，最大的动作

$$\pi'(s) \leftarrow \arg\max_{a} \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V^{\pi}(s') \right]$$

3.迭代循环

评估新策略π’ → 改进得到π”→ ….直到策略不再变化，此时π*即为最优策略

价值迭代

跳过显式策略，直接迭代最优价值V*

步骤：

1.初始化：任意设定$V_0(s)$

2.迭代更新：

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V_k(s')\right]$$

每次对所有状态和动作取最大值，逼近V*

3.收敛判断：设定阈值

4.提取策略：收敛后，最优策略为：

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

- 策略迭代：像"边学边改"——每学完一章调整学习方法。
- 价值迭代：像"先学完全部内容，再一次性总结最优方法"。

无模型强化学习

蒙特卡洛方法（Monte Carlo, MC）

MC方法让智能体完整经历一个回合（episode）后，根据实际获得的累计回报（如游戏总分）回溯评估每个状态或状态-动作对的价值。

步骤：

1.生成轨迹

用策略π运行一局游戏，记录轨迹：

$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T$$

2.计算回报G_t

对每个时间步 t，计算从 t 开始的折扣回报：

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$$

3.更新价值函数

- 对每个状态s_t首次出现的位置：
  - 将G_t加入样本集合
  - 更新$V^\pi(s_t)$为所有历史G_t的平均值：

$$V^\pi(s_t) \leftarrow \text{average}(G_t^{(1)}, G_t^{(2)}, \ldots)$$

4.重复：运行多个回合，逐步收敛到真实V^π

时序差分学习（Temporal Difference, TD）

TD方法让智能体每一步都学习，利用‘短期预测’（自举）更新价值，无需等待回合结束。

TD预测算法 → 评估一个给定的策略的价值函数

步骤：

1.初始化：任意初始化V(s)

2.单步交互：

在状态s_t，按策略选择动作a_t，得到奖励r_{t+1}和下一状态s_{t+1}

3.TD更新：

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

TD误差：$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ (现实与预测的差距)

4.重复：持续交互并更新，直至收敛。

时序差分学习（Temporal Difference, TD）

TD控制算法 → 找到最优策略

1.SARSA（State-Action-Reward-State-Action）

更新规则：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

核心特点：

- **同策略（On-Policy）**：
  - 使用**当前策略** π 选择下一步动作 a_{t+1}（如ε-greedy策略）。
  - 直接优化当前策略，学习过程中策略会逐步改进。
- **保守性**：
  - 更新时考虑实际执行的 a_{t+1}，若 a_{t+1} 有风险（如靠近悬崖），会降低 $Q(s_t, a_t)$ 的估值。

时序差分学习（Temporal Difference, TD）

2.Q-learning

更新规则：

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

(1) 同策略：边学边改

- **类比**：学生用自己的学习方法（策略）做练习题（探索），并根据做题结果（奖励）**调整同一套方法。**
- **特点**：学习过程保守，但可能陷入局部最优。

杉

(2) 异策略：旁观者清

- **类比**：学生观察其他同学（行为策略）的解题方法，但最终总结出一套**更优的自己的方法**（目标策略）。
- **特点**：可复用历史数据，探索更高效。
    - **可能高估**：
        - max 操作可能导致Q值过估计（可通过Double Q-learning改进）。

值函数近似

值函数近似(Value Function Approximation, VFA)

概念：值函数近似 (Value Function Approximation, VFA) 是一种在强化学习中用于估计状态价值函数 $V_\pi(s)$ 或动作价值函数 $Q_\pi(s,a)$ 的方法。与无模型强化学习中直接使用表格来存储每个状态（或状态-动作对）的值不同，值函数近似使用一个参数化的函数来逼近真实的值函数。

这个参数化的函数可以是各种形式，例如：

- **线性函数：** $\hat{v}(s,\mathbf{w}) = \mathbf{w}^\top \phi(s)$ 或 $\hat{q}(s,a,\mathbf{w}) = \mathbf{w}^\top \phi(s,a)$，其中 $\phi(s)$ 或 $\phi(s,a)$ 是状态或状态-动作的特征向量，$\mathbf{w}$ 是权重向量。
- **非线性函数：** 神经网络（深度学习）、决策树、核方法等。

这里的 $\mathbf{w}$ 是函数的参数，我们的目标是通过学习调整这些参数，使得近似值函数 $\hat{v}(s,\mathbf{w})$ 或 $\hat{q}(s,a,\mathbf{w})$ 尽可能地接近真实的价值函数。

原理：值函数近似的核心思想是利用泛化能力。通过学习状态（或状态-动作）的特征与对应值之间的关系，近似函数可以将学到的知识推广到未曾访问过的状态（或状态-动作对）上。

应用：值函数近似在解决具有大规模或连续状态和/或动作空间的强化学习问题中至关重要。比如游戏AI、机器人控制、自动驾驶、推荐系统等

谢谢聆听！