# Deep Q-Network

商秋林

2025/4/2

北京大學
PEKING UNIVERSITY

PART 01

# 时序差分方法

# 无模型的强化学习

有模型的强化学习：

- 需要构建环境的模型，通常指的是**状态转移概率 P(s′|s,a)** 和**奖励函数 R(s,a)**， 即给定一个状态 s 和动作 a，可以预测下一状态 s′ 和对应的奖励 R。
- 典型算法：动态规划（DP）

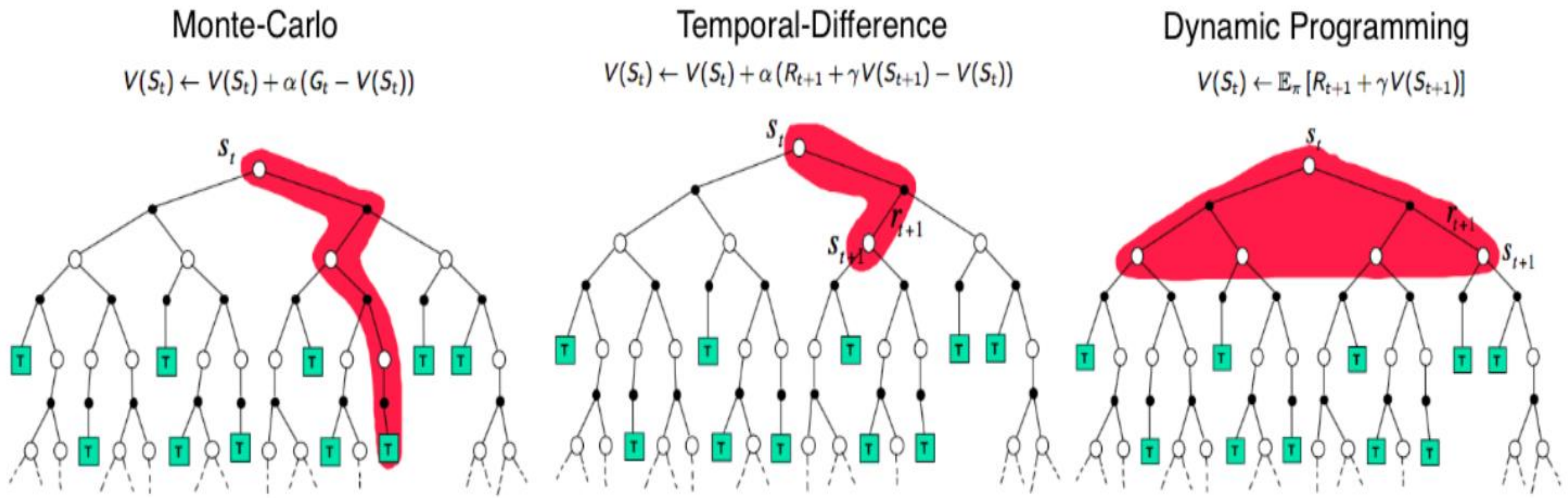
无模型的强化学习：

- 不需要显式建模环境，而是直接通过与环境交互来学习策略或价值函数。它不知道状态转移概率 P(s′|s,a) 和奖励函数 R(s,a)， 而是通过试探和估计来处理决策问题。
- 典型算法：时间差分（TD）方法，如 Q-Learning 和 SARSA，直接更新价值函数。

# 时序差分方法

时序差分方法用于估计策略的**价值函数**，结合了蒙特卡洛和动态规划的思想。

- 和蒙特卡洛的相似之处在于可以从样本数据中学习，不需要事先知道环境
- 和动态规划的相似之处在于根据贝尔曼方程的思想，利用后续状态的价值估计来更新当前状态的价值估计



# 时序差分方法

时序差分方法：

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

- 时序差分算法用**当前获得的奖励**加上**下一个状态的价值**估计来作为在当前状态会获得的回报
- 时序差分vs蒙特卡洛

| | 计算时机 | 回报计算方式 |
|---|---|---|
| 蒙特卡洛 | 整个序列结束后 | 回报 = 整个序列的累计奖励 |
| 时序差分 | 当前步结束即可 | 回报 ≈ 当前奖励 + 下一个状态的价值估计 |

# 为什么 TD目标 能替代 Gt？

$$
\begin{aligned}
V_\pi(s) &= \boxed{\mathbb{E}_\pi[G_t \mid S_t = s]} \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_t + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \\
&= \boxed{\mathbb{E}_\pi[R_t + \gamma V_\pi(S_{t+1}) \mid S_t = s]}
\end{aligned}
$$

# PART 02

## SARSA & Q-learning

# SARSA

SARSA算法用于估计策略 $\pi$ 下的动作价值函数$Q_\pi(S,a)$

$$Q_\pi(s_t,a_t) \leftarrow Q_\pi(s_t,a_t) + \alpha[r_t + \gamma Q_\pi(s_{t+1},a_{t+1}) - Q_\pi(s_t,a_t)]$$

动作的选择使用ϵ-贪婪策略

$$\pi(a \mid s) = \begin{cases} \epsilon/|\mathcal{A}| + 1 - \epsilon & \text{如果 } a = \arg\max_{a'} Q(s,a') \\ \epsilon/|\mathcal{A}| & \text{其他动作} \end{cases}$$

# SARSA 实际过程

北京大学 PEKING UNIVERSITY

**SARSA（State-Action-Reward-State-Action）**

1. **动作** $a_t$：
   - 在当前状态 $s_t$，使用 $\epsilon$-**贪婪策略** 选择动作 $a_t$。
     - 以概率 $1-\epsilon$ 选择当前 $Q(s_t, a)$ 中最大值对应的动作（利用）。
     - 以概率 $\epsilon$ 随机选择一个动作（探索）。
2. **动作** $a_{t+1}$：
   - 在下一状态 $s_{t+1}$，**再次使用** $\epsilon$-**贪婪策略** 选择动作 $a_{t+1}$。
   - $a_{t+1}$ 是根据当前行为策略选择出来的动作，而不是直接取 $Q(s_{t+1}, a)$ 的最大值，因此体现了 **on-policy** 的特性。
3. **更新** $Q(s_t, a_t)$：

   SARSA 的更新公式为：

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

   - 更新时依赖实际选择的 $a_{t+1}$，这是 SARSA 的关键。

| **动作** $a$ | $Q(s_t, a)$ |
|---|---|
| 上（up） | 5 |
| 下（down） | 10 |
| 左（left） | 0 |
| 右（right） | 7 |

# Q-learning

Q-learning 算法用于估计策略 $\pi$ 下的动作价值函数$Q_\pi(S,a)$

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

动作的选择使用ϵ-贪婪策略：

$$\pi(a \mid s) = \begin{cases} \epsilon/|\mathcal{A}| + 1 - \epsilon & \text{如果 } a = \arg\max_{a'} Q(s,a') \\ \epsilon/|\mathcal{A}| & \text{其他动作} \end{cases}$$

# Q-learning 实际过程

## Q-Learning

1. **动作** $a_t$：
   - 在当前状态 $s_t$，使用 $\epsilon$-**贪婪策略** 选择动作 $a_t$。
     - 以概率 $1-\epsilon$ 选择当前 $Q(s_t, a)$ 中最大值对应的动作。
     - 以概率 $\epsilon$ 随机选择一个动作。
2. **动作** $a_{t+1}$：
   - 在下一状态 $s_{t+1}$，直接选择 $Q(s_{t+1}, a)$ 中的最大值对应的动作（即 $\arg\max_a Q(s_{t+1}, a)$），而不使用 $\epsilon$-贪婪策略。
   - 不考虑实际行为策略，而是更新时假设总是采取最优动作，因此体现了 **off-policy** 的特性。
3. **更新** $Q(s_t, a_t)$：

   Q-Learning 的更新公式为：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

   - 更新时基于下一状态中最优动作的价值（$\max_{a'} Q(s_{t+1}, a')$）。

| **动作** $a$ | $Q(s_t, a)$ |
|---|---|
| 上（up） | 5 |
| 下（down） | 10 |
| 左（left） | 0 |
| 右（right） | 7 |

# SARSA vs Q-learning

| | SARSA | Q-learning |
|---|---|---|
| 实际动作 | ϵ-贪婪策略 | ϵ-贪婪策略 |
| 评估动作 | ϵ-贪婪策略 | 直接选择 $argmax_a(S_{t+1}, a)$ |
| 策略类型 | on-policy | off-policy |
| 目标值计算 | $r_t + \gamma Q(s_{t+1}, a_{t+1})$ | $r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$ |
| 适用场景 | 更新稳定，适合探索性较强的场景 | 收敛快，适合需要快速找到最优策略的场景 |

关于On Policy和Off Policy的区别

PART 03

# DQN 算法

# 从表格到函数

北京大学 PEKING UNIVERSITY

Q-learning 算法中，我们以表格的形式存储动作价值函数Q，这种用表格存储动作价值的做法只在环境的状态和动作都是离散的，并且空间都比较小的情况下适用

当状态或者动作数量非常大的时候，这种做法就不适用了。例如，当状态是一张 RGB 图像时，假设图像大小210×160×3是，此时一共有种$256^{210\times160\times3}$状态，在计算机中存储这个数量级的值表格是不现实的。更甚者，当状态或者动作连续的时候，就有无限个状态动作对，我们更加无法使用这种表格形式来记录各个状态动作对的值。

对于这种情况，我们需要用函数拟合的方法来估计Q值，即将这个复杂的值表格视作数据，使用一个参数化的函数$Q_\theta$来拟合这些数据

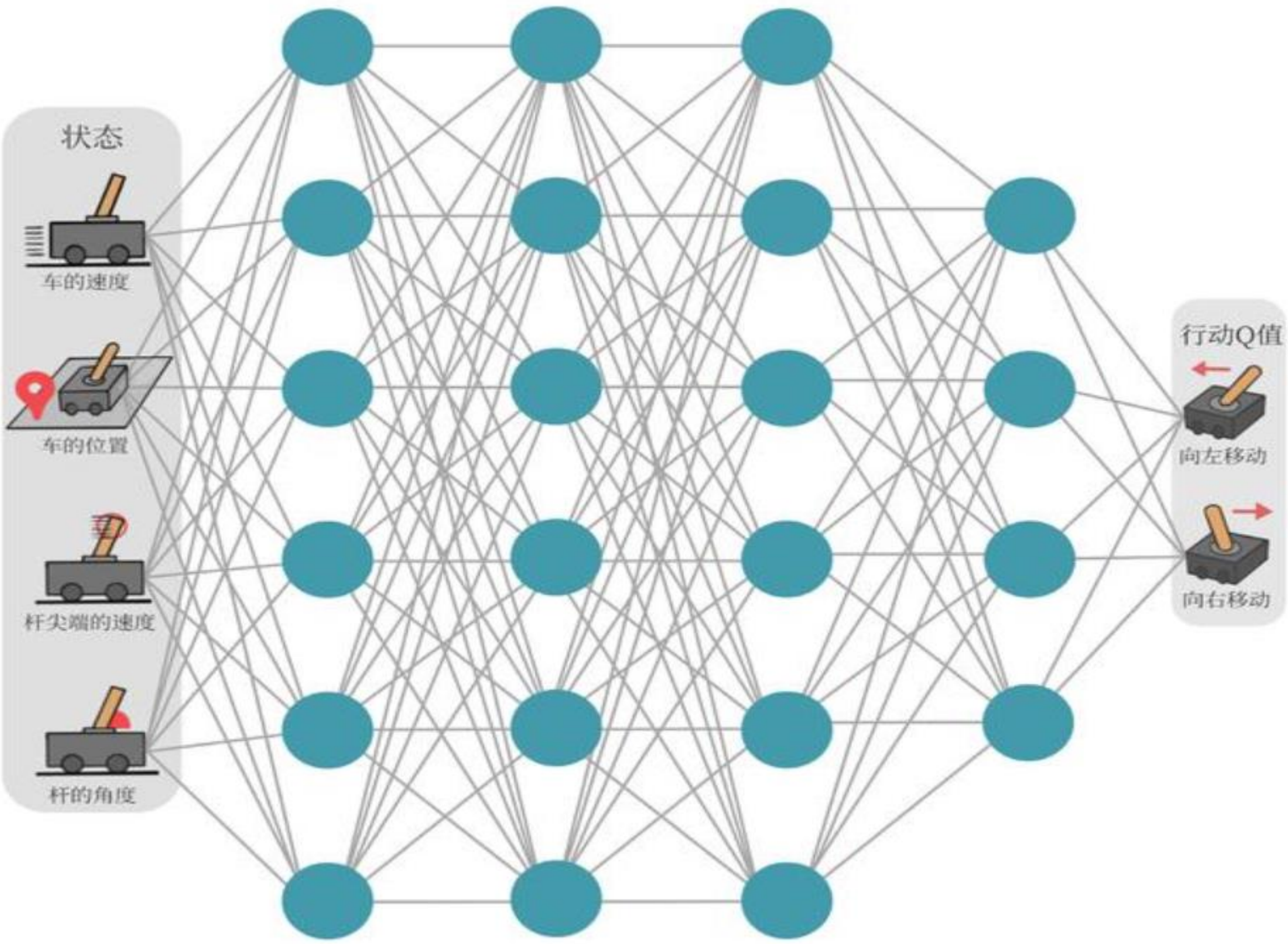
# 从表格到函数

我们可以用一个神经网络来表示函数

- 若动作是连续（无限）的，神经网络的输入是状态s和动作a，然后输出一个标量，表示在状态s下采取动作a能获得的价值
- 若动作是离散（有限）的，除了可以采取动作连续情况下的做法，我们还可以只将状态s输入到神经网络中，使其同时输出每一个动作的Q值

通常 DQN（以及 Q-learning）只能处理动作离散的情况，因为在函数的更新过程中有maxQ(s,a)这一操作

# CartPole 环境中的 Q 网络示意图



# Q 网络的损失函数

回顾 Q-Learning 的更新规则

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ \boxed{r_t + \gamma \max_{a'} Q\left(s_{t+1}, a'\right)} - Q(s_t, a_t) \right]$$

核心思想：利用 **TD 目标**（Temporal Difference）来逼近真实的 Q 值。

构造损失函数：最小化 Q 值的估计与 TD 目标之间的均方误差（MSE）

$$L(\omega) = \frac{1}{2N} \sum_{i=1}^{N} \left[ Q_\omega(s_i, a_i) - \left( r_i + \gamma \max_{a'} Q_\omega\left(s_i', a'\right) \right) \right]^2$$

其中 N 表示小批量样本的数量，用于计算平均损失

# 关键技术：经验回放

1. 我们在与环境交互时，会收集很多数据（状态、动作、奖励、下一状态等）
2. 不是直接用这些数据更新 Q 网络，我们把这些数据存储在一个经验回放缓冲区（Replay Buffer）中
3. 在训练时，从缓冲区中 **随机采样** 一部分数据来更新网络，而不是直接使用最新采集的数据

# 关键技术：经验回放



经验回放的作用：

1.打破样本间的相关性，满足独立同分布假设

2.提高数据使用效率

3.平滑学习过程，减少策略震荡

# 关键技术：目标网络

DQN 的目标是让 $Q_\omega(s,a)$接近 TD 目标 $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$

**问题**：TD 目标依赖于神经网络的输出，若训练网络参数不断更新，目标值也会不断变化，导致训练过程不稳定。

**解决方法**：通过引入目标网络 $Q_{\omega^-}(s,a)$，将 TD 目标值固定一段时间，减少目标值的波动。

**实现方式**：

- 在训练中，使用目标网络 $Q_{\omega^-}(s,a)$来计算 TD 目标，只更新训练网络 $Q_\omega(s,a)$
- 目标网络的参数 ω- 每隔若干步与训练网络的参数 ω 同步一次

**TD目标**：

$$r + \gamma \max_{a'} Q_{\omega^-}(s_{t+1}, a')$$

# 关键技术：目标网络



目标网络的好处：

1. 减少目标值的波动，稳定训练过程
2. 防止发散，减少更新的相互影响
3. 缓解过估计问题

北京大學 PEKING UNIVERSITY

PART 04

# DQN 改进算法

# Double DQN

普通的 DQN 算法仍然会导致对价值的过估计（overestimation）

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_t + \boxed{\gamma \max_{a'} Q_{\omega^-}\left(s_{t+1}, a'\right)} - \boxed{Q(s_t, a_t)} \right]$$

其中，max操作会选择下一状态中Q值最大的动作。然而，由于以下原因，Q值的估计可能被高估：

1. 估计误差：神经网络对Q值的预测可能存在噪声。

2. 最大化偏差：max操作会放大误差，即使某些动作的真实Q值较低，其估计值可能因噪声被误认为更高。

# Double DQN

| Action | 预测Q值 | 真实Q值 |
| --- | --- | --- |
| a1 | 0.9 | 1.0 |
| a2 | 1.2 | 0.5 |

带来两个问题：

1. 训练时用到的Q值可能被高估
2. 可能选择次优动作

过估计问题会随着训练逐步放大：

1. 误差传播：高估的目标Q值会通过贝尔曼方程传播到更早的状态。
2. 策略偏移：策略倾向于选择被高估的动作，但这些动作的实际收益可能较低，导致次优策略。

# Double DQN

Double DQN 的TD目标：

$$r + \gamma Q_{\omega^-}\left(s_{t+1}, \arg\max_{a'} Q_{\omega}(s_{t+1}, a')\right)$$

1. **动作选择** 使用训练网络$Q_w$而不是目标网络 $Q_{w^-}$

2. **Q 值估计** 依然由目标网络 $Q_{w^-}$ 完成，但只计算训练网络选择的动作 a* 的值。

# Double DQN

为什么 Double DQN 能大幅度改善过估计的问题?

$$r + \gamma Q_{\omega^-}\left(s_{t+1}, \arg\max_{a'} Q_\omega(s_{t+1}, a')\right)$$

1. 从直觉上：避免目标网络既选择动作又计算价值，造成高估

2. 从数学上：由于 a* 是训练网络选择的动作，不一定是目标网络中 Q 值最大的动作，Q值估计会更保守，从而减少过高估计

$$\underbrace{Q\left(s_{j+1}, a^\star; \boldsymbol{w}^-\right)}_{\text{双 Q 学习}} \leq \underbrace{\max_{a\in\mathcal{A}} Q\left(s_{j+1}, a; \boldsymbol{w}^-\right)}_{\text{用目标网络的 Q 学习}}.$$

# Dueling DQN

在传统DQN中，网络直接输出每个动作的Q值Q(s,α)，但这种方式存在一个关键问题：某些状态下，所有动作的价值主要由状态本身决定，而非动作之间的差异。例如：

- 赛车游戏中：如果前方是悬崖，无论左转还是右转（动作），只要避开悬崖，状态的整体价值（存活）都很高，此时动作之间的差异较小。
- 导航任务中：如果智能体处于安全区域，所有动作（前进、左转、右转）的长期收益可能相近，但若处于危险区域（如靠近敌人），不同动作的收益差异会显著增大。

问题本质：直接学习Q(s,α)会导致网络重复建模状态的价值（例如“安全"或”危险")，而这些信息对所有动作是共享的。这会降低学习效率，尤其是在动作对结果影响较小的场景中。

# Dueling DQN

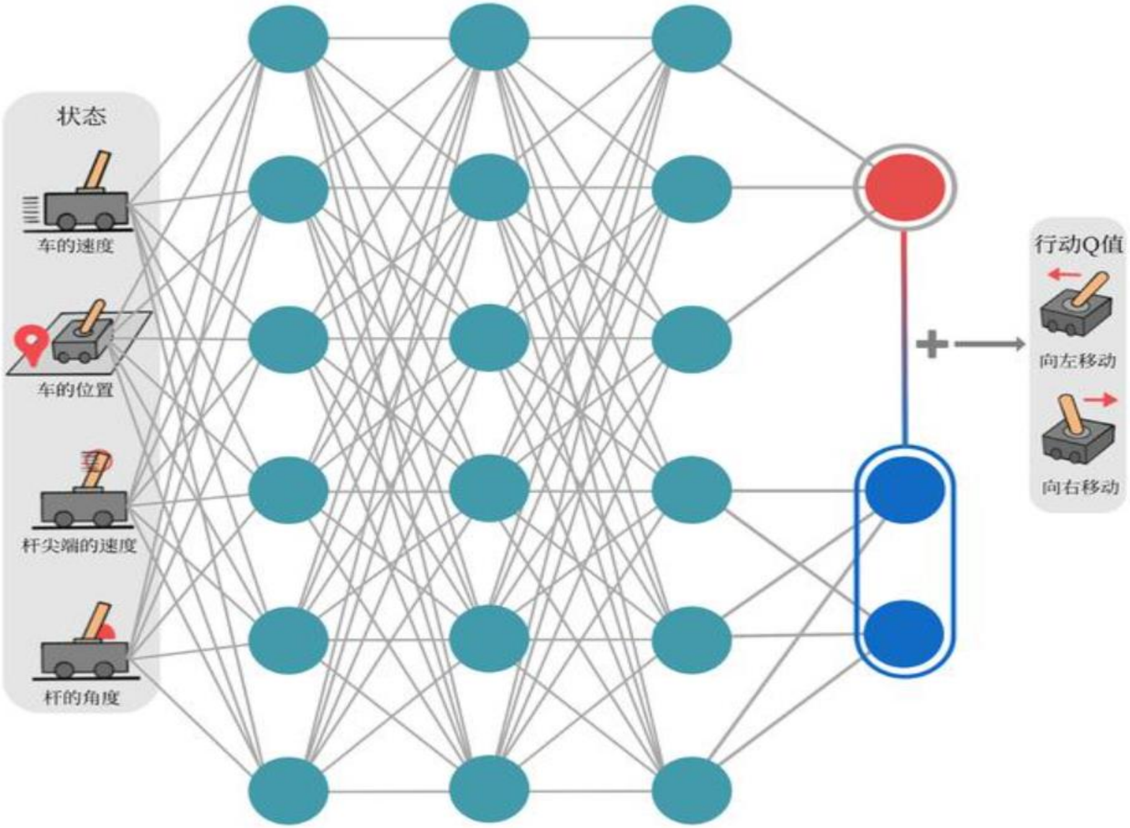
Dueling DQN 将Q分解为两个部分：

$$Q(s, a) = V(s) + A(s, a)$$

- 状态价值函数 V(s)：反映状态 s 的总体价值，与具体动作无关。
- 优势函数 A(s,a)：表示动作 a 相对于其他动作的好坏程度。

# Dueling DQN

在这样的模型下，我们不再让神经网络直接输出Q值，而是训练神经网络的最后几层的两个分支，分别输出状态价值函数和优势函数，再求和得到Q值



# Dueling DQN

$Q(s,a) = V(s) + A(s,a)$存在对于V值和A值建模不唯一性的问题：

- 对于同一组 Q(S,a) ,存在无数种 V(s) 和 A(s,a) 的组合。
- 例如：若将 V(s) 增加一个常数 C ,同时将所有 A(s,a) 减少 C ,则 Q(s,a) 保持不变。

这种不唯一性会导致两个严重问题：

1. V 和 A 的 “职责 “不明确：Q 网络不知道 V(s) 表示状态价值，A(s,a) 表示动作优势

2.训练不稳定：梯度更新时，网络可能同时调整 V(s) 和 A(s,a) 的值，导致它们的值剧烈波动，但Q(s,a) 却保持稳定。这会降低训练效率。

# Dueling DQN

为了消除上述不唯一的问题，Dueling DQN 通过约束 A(s,a) 的值，确保动作优势的实际输出在特定条件下为 0。

1. $$Q(s,a) = V(s) + \left(A(s,a) - \max_{a'} A(s,a')\right)$$

   此时V(s)的实际意义为Q的最大值：$V(s) = Q(s,a^*) = \max_{a'} Q(s,a')$

2. $$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a')\right)$$

   此时V(s)的实际意义为Q的平均值：$V(s) = \frac{1}{|\mathcal{A}|}\sum_{a'} Q(s,a')$

# 优先经验回放

在传统DQN中，经验回放(Experience Replay)通过存储历史经验(s,a,r,s’)并**均匀随机采样**来训练网络。然而，均匀采样忽略了不同经验的重要性差异。例如：

- 某些经验（如高TD误差的样本）可能包含更多有价值的信息，值得被更频繁地学习。
- 某些经验（如稀疏奖励场景中的成功轨迹）可能对策略改进至关重要，但出现概率极低。

优先经验回放的核心思想：根据经验的重要性赋予不同的采样概率，优先回放对学习更有帮助的样本，从而加速收敛并提升性能。

# 优先级度量：TD 误差



优先级的依据是时序差分误差（TD Error），定义为：

$$\delta = \left| r + \gamma \max_{a'} Q_{\text{target}}\left(s', a'\right) - Q(s, a) \right|$$

- TD 误差大：网络预测值与目标值差距大，说明该样本尚未被充分学习。
- TD 误差小：样本已被较好学习，优先级较低。

# 两种优先级分配方式

1. 基于比例的优先级（Proportional Prioritization）：

$$p_i = |\delta_i| + \epsilon$$

- $\epsilon$>0 是极小常数，避免零优先级。
- 采样概率与 pi 成正比。

2. 基于排序的优先级（Rank-based Prioritization）：

$$p_i = \frac{1}{\mathrm{rank}(|\delta_i|)}$$

# 采样策略



样本 i 的采样概率为：

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha}$$

- α ∈[0,1]： 控制优先级的强度
- α =0：退化为均匀采样
- α =1：完全按优先级采样。

# 重要性采样

我们通过最小化TD 误差的平方（即损失函数）来更新 Q 网络：

$$\mathcal{L} = \mathbb{E}_{(s,a,r,s')\sim \text{经验池}}\left[\delta^2\right], \quad \delta = r + \gamma \max_{a'} Q_{\text{target}}\left(s', a'\right) - Q(s, a).$$

- 均匀采样：每个样本被采样的概率为 $P_{\text{均匀}}(i) = \frac{1}{N}$（N 是经验池大小）

$$\nabla \mathcal{L}_{\text{均匀}} = \mathbb{E}_{\text{均匀}}\left[\nabla \delta^2\right] = \frac{1}{N}\sum_{i=1}^{N} \nabla \delta_i^2.$$

- 优先采样：每个样本被采样的概率 $P(i)$ 与其 TD 误差正相关

$$\nabla \mathcal{L}_{\text{优先}} = \mathbb{E}_{P(i)}\left[\nabla \delta^2\right] = \sum_{i=1}^{N} P(i) \nabla \delta_i^2.$$

# 重要性采样

为了修正梯度方向，需定义重要性采样比将优先采样的梯度转换为均匀采样的等效形式：

$$\rho_i = \frac{P_{\text{均匀}}(i)}{P(i)} = \frac{1/N}{P(i)}$$

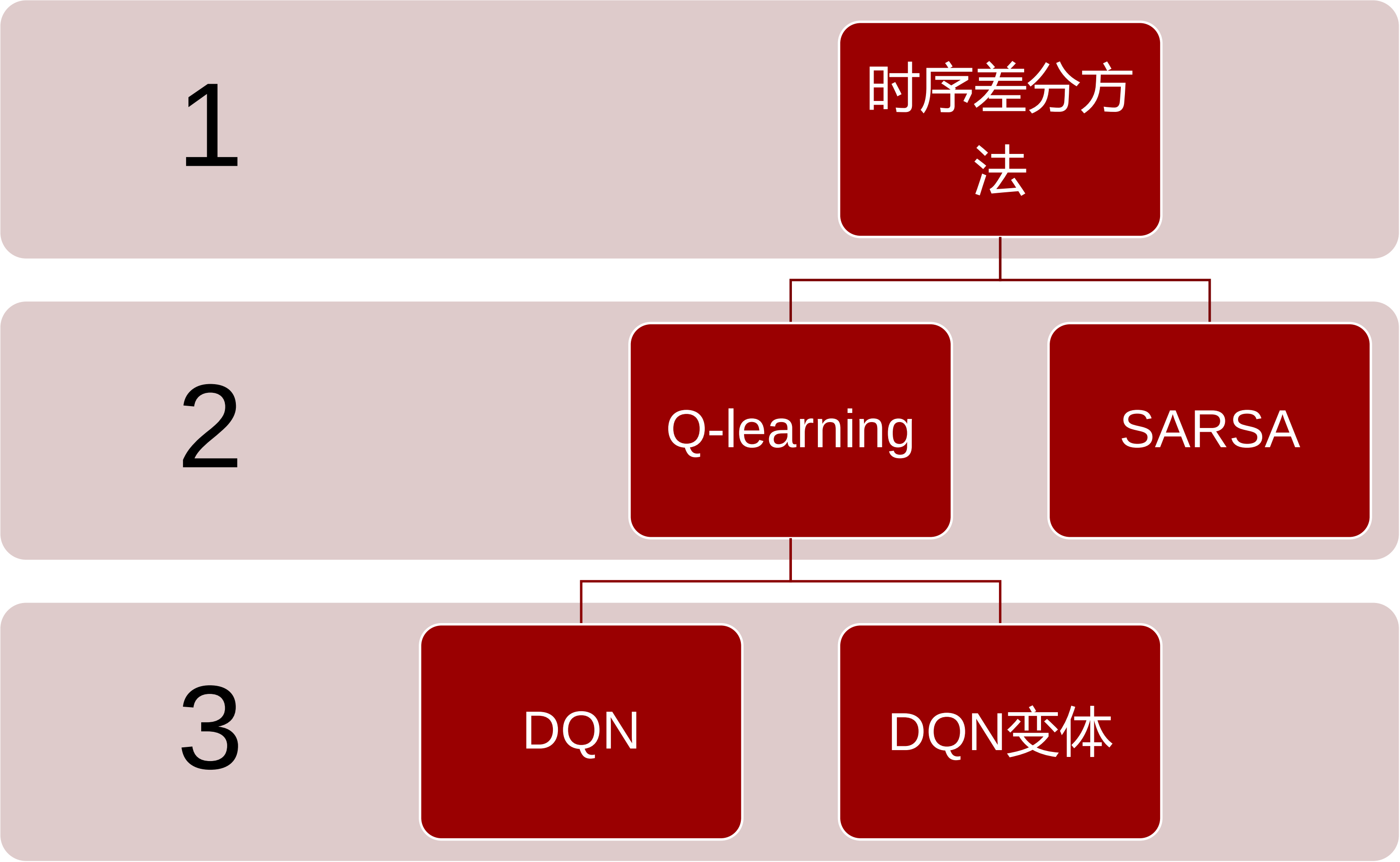
此时，优先采样的梯度可重写为：

$$\nabla \mathcal{L}_{\text{优先}} = \mathbb{E}_{P(i)}\left[\rho_i \nabla \delta_i^2\right] = \mathbb{E}_{\text{均匀}}\left[\nabla \delta_i^2\right]$$

定义重要性采样权重并代入损失函数：

$$w_i = \rho_i^\beta = \left(\frac{1}{NP(i)}\right)^\beta \Rightarrow \mathcal{L} = \frac{1}{B}\sum_{i=1}^{B} w_i \delta_i^2$$

# 小结



北京大學
PEKING UNIVERSITY

# Thank you!