南京大学
人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE, NANJING UNIVERSITY

第五部分：　多Agent强化学习

章宗长

2023年5月23日

内容安排

单Agent强化学习

- **强化学习的基本设定**
- 动态规划
- 基于值函数的强化学习
- 基于策略梯度的强化学习

不同类型学习的反馈信号

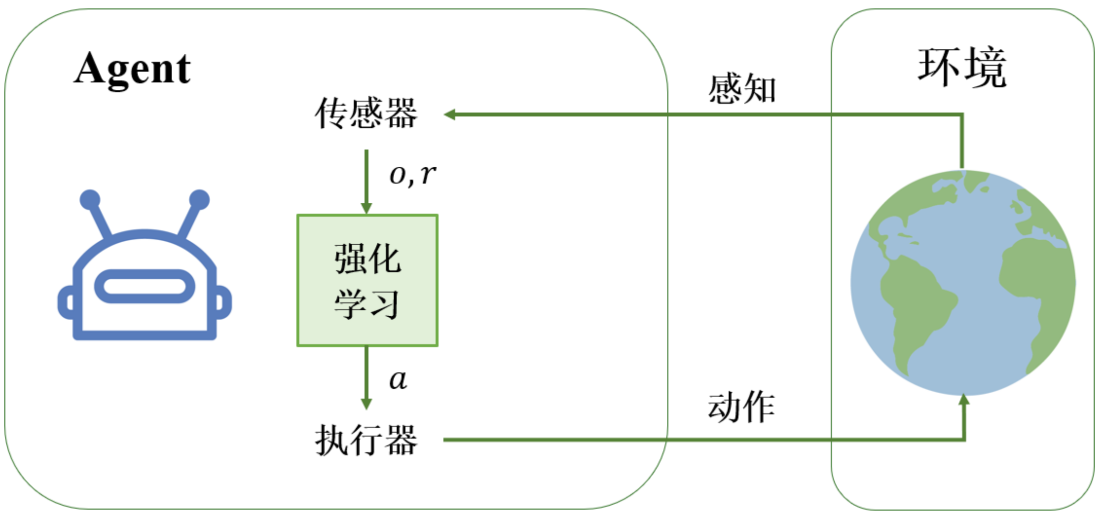
- 监督学习
  - 环境提供形如“特征-标记”的教师信号
  - 明确指出策略的输入与输出
- 无监督学习
  - 环境只需要提供形如“特征”的训练信息
  - 没有指定策略的输出
- 强化学习
  - 环境提供的是Agent动作好坏的一种评价
    - 通常为形如“奖励/惩罚”的标量信号
  - 仅仅隐含了策略的最优输出，而没有告知正确的输出

强化学习任务及其挑战

- **强化学习**
  - 环境模型未知
  - 从经验中学习如何行动
    - 通过观察动作的结果，选择最大化期望累积奖励的动作
- **挑战**
  - 探索与利用（exploration and exploitation）
    - 在探索环境和利用从经验中获得的知识之间保持平衡
  - 信度分配（credit assignment）
    - 奖励具有延迟性
  - 泛化（generalization）
    - 从有限的经验中获得可泛化的策略

标准的强化学习模型

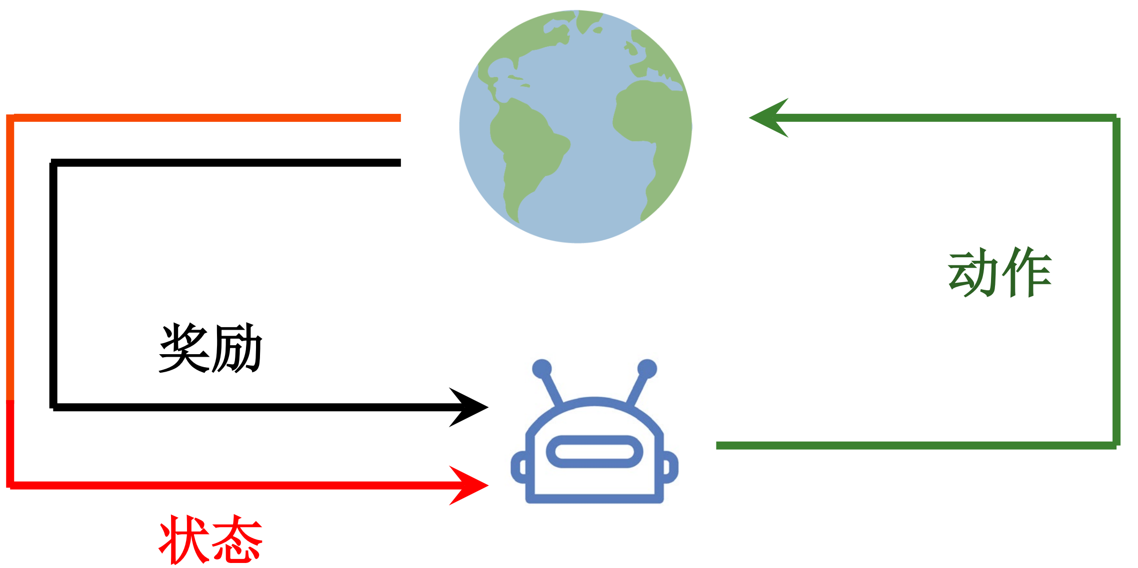
- Agent与环境交互的每一步：
  - 感知环境的状态$s \in \mathcal{S}$，得到观察$o \in \mathcal{O}$
  - 根据观察做出动作$a \in \mathcal{A}$
  - 环境给予Agent奖励$r \in R$，并进入下一步的状态$s' \in \mathcal{S}$



- 两个困难
  - 相邻步交互的时间间隔不固定
  - Agent不一定能准确感知环境的状态
    - $o \neq s$, $\mathcal{O} \neq \mathcal{S}$

简化后的交互模型

- 在离散时刻$t = 0, 1, 2, \ldots$，Agent与环境的交互过程：
  - 准确感知环境的状态$S_t = s$
  - 根据状态做出动作$A_t = a$
  - 环境给予Agent奖励$R_t = r$，并进入下一步的状态$S_{t+1} = s'$



Agent-环境交互的示例

- $t = 0$

你在状态1处。你有4个可选动作。

我执行动作2。

- $t = 1$

你得到了+3的奖励，正处在状态5，有2个可选动作。

我执行动作1。

- $t = 2$

你得到了-5的奖励，正处在状态1，有4个可选动作。

我执行动作2。

…… ……

马尔可夫假设

- 给定过去时刻的轨迹$S_0, A_0, R_0, \dots, S_{t-1}, A_{t-1}, R_{t-1}$, S_t, A_t，状态S_{t+1}和奖励R_t的概率分布为：

$$\mathrm{P}(S_{t+1} = s', R_t = r \mid S_0, A_0, R_0, \dots, S_{t-1}, A_{t-1}, R_{t-1}, S_t, A_t)$$

- 马尔可夫假设：状态S_{t+1}和奖励R_t仅依赖于当前状态S_t和动作A_t，与更早的状态和动作无关：

$$\mathrm{P}(S_{t+1} = s', R_t = r \mid S_t = s, A_t = a)$$

马尔可夫决策过程

- 马尔可夫决策过程（Markov Decision Process, MDP）
  - 状态空间$\mathcal{S}$
  - 动作空间$\mathcal{A}$
  - 奖励空间$\mathcal{R}$
  - 动力（dynamics）函数

$$P(S_{t+1}, R_t \mid S_t, A_t): \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \longrightarrow [0,1]$$

$$\Downarrow$$

状态转移函数：$P(S_{t+1} \mid S_t, A_t) = \sum_{r \in \mathcal{R}} P(S_{t+1}, R_t = r \mid S_t, A_t)$

奖励函数：$P(R_t \mid S_t, A_t) = \sum_{s' \in \mathcal{S}} P(S_{t+1} = s', R_t \mid S_t, A_t)$

稳态MDPs

- $P(S_{t+1}, R_t \mid S_t, A_t)$ 不随时间发生变化
- 动力函数 $P(s', r \mid s, a) = P(S_{t+1} = s', R_t = r \mid S_t, = s\ A_t = a)$
- $P(S_{t+1} \mid S_t, A_t)$和$P(R_t \mid S_t, A_t)$不随时间发生变化
- 状态转移函数

$$P(s' \mid s, a) = P(S_{t+1} = s' \mid S_t = s, A_t = a) = \sum_{r \in \mathcal{R}} P(s', r \mid s, a)$$

- 给定“状态-动作”的期望奖励函数

$$R(s, a) = \sum_{r \in \mathcal{R}} r \cdot P(r \mid s, a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r \mid s, a)$$

策略

- 策略$\pi_t(h_t)$：给定历史$h_t = (s_{0:t}, a_{0:t-1})$，确定动作
- 一个MDP的策略$\pi_t(s_t)$
  - 未来的状态和奖励仅依赖于当前状态和动作
- 一个稳态MDP的随机性策略$\pi: \mathcal{S} \times \mathcal{A} \longrightarrow [0,1]$可以定义为

$$\pi(a \mid s) = \mathrm{P}(A_t = a \mid S_t = s), \qquad s \in \mathcal{S}, a \in \mathcal{A}$$

- 一个稳态MDP的确定性策略$\pi: \mathcal{S} \longrightarrow \mathcal{A}$，即$\pi: s \longmapsto \pi(s)$

值函数

- 状态值函数$V^{\pi}(s)$：从状态s起，执行策略π的期望回报

$$V^{\pi}(s) = \mathbb{E}_{a_t \sim \pi(s_t)}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s\right]$$

贝尔曼期望等式 $$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V^{\pi}(s')$$

- 动作值函数$Q^{\pi}(s,a)$：在状态s采取动作a后，执行策略π的期望回报

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V^{\pi}(s')$$

贝尔曼期望等式 $$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) Q^{\pi}(s', \pi(s'))$$

最优策略

- 最优策略$\pi^*(a \mid s)$满足：

$$\pi^*(a \mid s) = \begin{cases} 1, & a \in \arg\max_{a' \in \mathcal{A}} Q^*(s, a') \\ 0, & \text{其他} \end{cases}$$

- 对于一个动力函数而言，可能存在多个最优策略

- 最优的确定性策略
  - 对所有$s \in \mathcal{S}$，$\pi^*(s) \in \arg\max_{a' \in \mathcal{A}} Q^*(s, a')$
  - 如果有多个动作a使得$Q^*(s, a)$取最大值，则任选一个动作即可

最优值函数

- 最优状态值函数$V^*(s)$：从状态s起，执行最优策略π^*的期望回报

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

贝尔曼最优等式 $V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^*(s') \right]$

- 最优动作值函数$Q^*(s,a)$：在状态s采取动作a后，执行最优策略π^*的期望回报

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^*(s')$$

贝尔曼最优等式 $Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \max_{a'} Q^*(s,a')$

单Agent强化学习

- 强化学习的基本设定
- **动态规划**
- 基于值函数的强化学习
- 基于策略梯度的强化学习

动态规划与贝尔曼等式

- **动态规划的要素**
  - 最优子结构：把原问题分解成多个子问题
  - 重叠子问题：重复利用子问题的解

- 贝尔曼期望等式 $V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V^\pi(s')$

- 贝尔曼最优等式 $V^*(s) = \max_a \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V^*(s') \right]$

- MDPs：（最优子结构）贝尔曼等式提供了递归地分解问题的方法，（重叠子问题）值函数存储的数据可以被重复使用
  - 策略迭代：用迭代的方式求解**贝尔曼期望等式**
  - 值迭代：用迭代的方式求解**贝尔曼最优等式**

值迭代

- 对**无限步数**的问题，最优状态值函数满足贝尔曼最优等式：

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^*(s') \right]$$

- 计算带折扣的k步最优值函数V_k：
  - 如果$k=0$，则对所有s：$V_0(s) \longleftarrow$任意值
  - 迭代地计算V_{k+1}：

$$V_{k+1}(s) \leftarrow \max_a \underline{Q_k(s,a)}$$

$$\boxed{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_k(s')}$$

- 常用的终止条件：

$$\underline{\|V_{k+1} - V_k\|_\infty} < \epsilon$$

贝尔曼残差

算法 1.1 值迭代算法

- **for** $s \in S$ **do**
  - 初始化 $V_0(s)$ 为任意值
- **end for**
- **for** $k = 0, 1, 2, \cdots$ **do**
  - **for** $s \in S$ **do**
    - **for** $a \in A$ **do**
      - $Q_k(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_k(s')$
    - **end for**
    - $V_{k+1}(s) \leftarrow \max_a Q_k(s,a)$
  - **end for**
  - 如果满足迭代终止条件，则跳出循环
- **end for**
- **for** $s \in S$ **do**
  - $\pi(s) \leftarrow \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_{k+1}(s') \right]$
- **end for**
- **return** π

策略迭代

- 策略评估：估计一个策略的期望回报

贝尔曼期望等式 $$V^{\pi}(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,\pi(s)) V^{\pi}(s')$$

- 策略改进：使用当前策略的值函数，计算一个新策略

算法 1.2 策略迭代算法

```
将策略 π_0 初始化为任意的确定性策略
for k = 0, 1, 2, ··· do
    for s ∈ S do
        V^{π_k}(s) ← R(s, π_k(s)) + γ ∑_{s'∈S} P(s' | s, π_k(s)) V^{π_k}(s')    ▷ 策略评估
    end for
    for s ∈ S do
        π_k(s) ← arg max_a R(s, a) + γ ∑_{s'∈S} P(s' | s, a) V^{π_k}(s')    ▷ 策略改进
    end for
    如果满足迭代终止条件 π_{k+1} = π_k，则跳出循环
end for
return π_k
```

单Agent强化学习

- 强化学习的基本设定
- 动态规划
- **基于值函数的强化学习**
- 基于策略梯度的强化学习

Q学习：异策略（off-policy）的TD控制

- 把增量估计用于动作值的贝尔曼最优方程：

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

$\Downarrow$ 增量估计

Q学习：使用最大化Q值的动作来更新Q值

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \Big[R_t + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\Big]$$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$

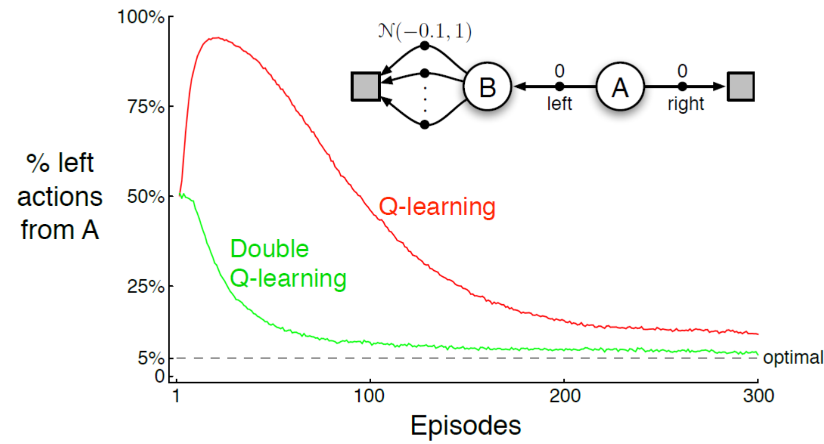
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., $\boxed{\varepsilon\text{-greedy}}$)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \boxed{\max_a Q(S',a)} - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

用于动作选择的策略是ε –贪心策略

用于评价Q值的策略是贪心策略

异策略（off-policy）：用于评价Q值的策略和用于动作选择的策略不同

双重Q学习及变种



- Q学习存在Q值高估的问题
  - 后果：学习最优策略的过程偏慢
  - 根源：用同一套Q值选择动作和评估所选动作的值

- 双重Q学习
  - 想法：存储两套Q值
    - 用一套选择动作
    - 用另一套评估所选动作的值
  - 不会高估Q值
  - 但可能低估Q值

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

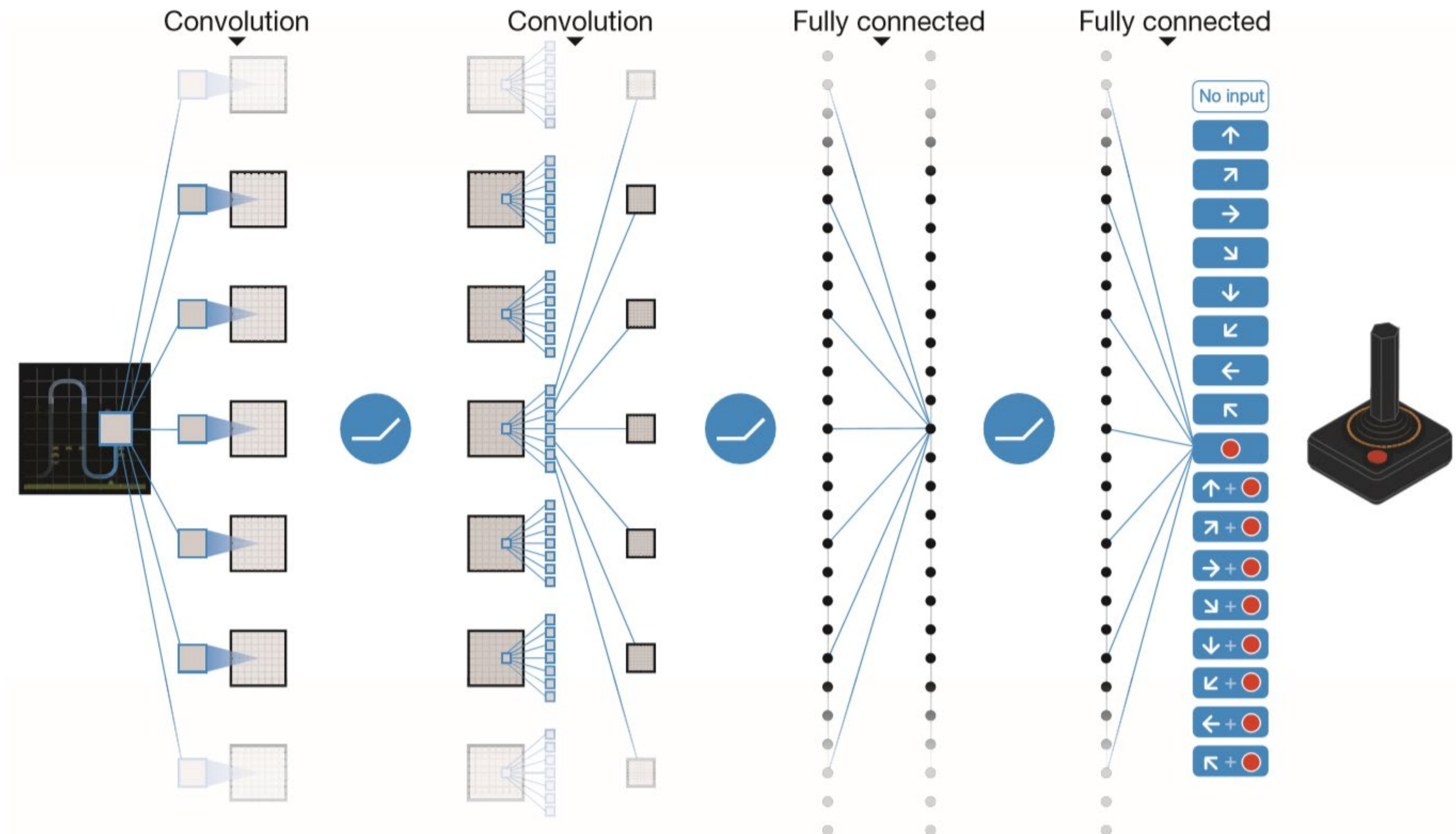
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma \underline{Q_2\big(S', \arg\max_a Q_1(S', a)\big)} - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma \underline{Q_1\big(S', \arg\max_a Q_2(S', a)\big)} - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until S is terminal

深度Q网络（Deep Q-Network，DQN）

- 第一个有广泛影响力的深度强化学习系统
  - 使用同一个结构的神经网络，学习玩多款不同的Atari 2600街机游戏
  - 在大多数游戏上，能达到或超越人类玩家的水平
- Q学习+卷积神经网络
- 经验回放
  - 将经验元组(s, a, s', r)存储起来，再按一定的规则从中采样
- 两个结构完全相同的Q网络
  - **评价网络**：计算回报的预测值，每次迭代都更新其权重
  - **目标网络**：作为学习的目标，用于解决训练不稳定的问题

目标网络是不会在每次迭代都变化，是一个固定的目标

在完成一定次数的更新后，再将评价网络的权重赋给目标网络，进而进行下一批更新


Convolution
Convolution
Fully connected
Fully connected
No input

DQN的网络结构

Algorithm 1: deep $\boxed{\text{Q-learning}}$ with experience replay.

免模型、异策略

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

ε-贪心

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

经验回放

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

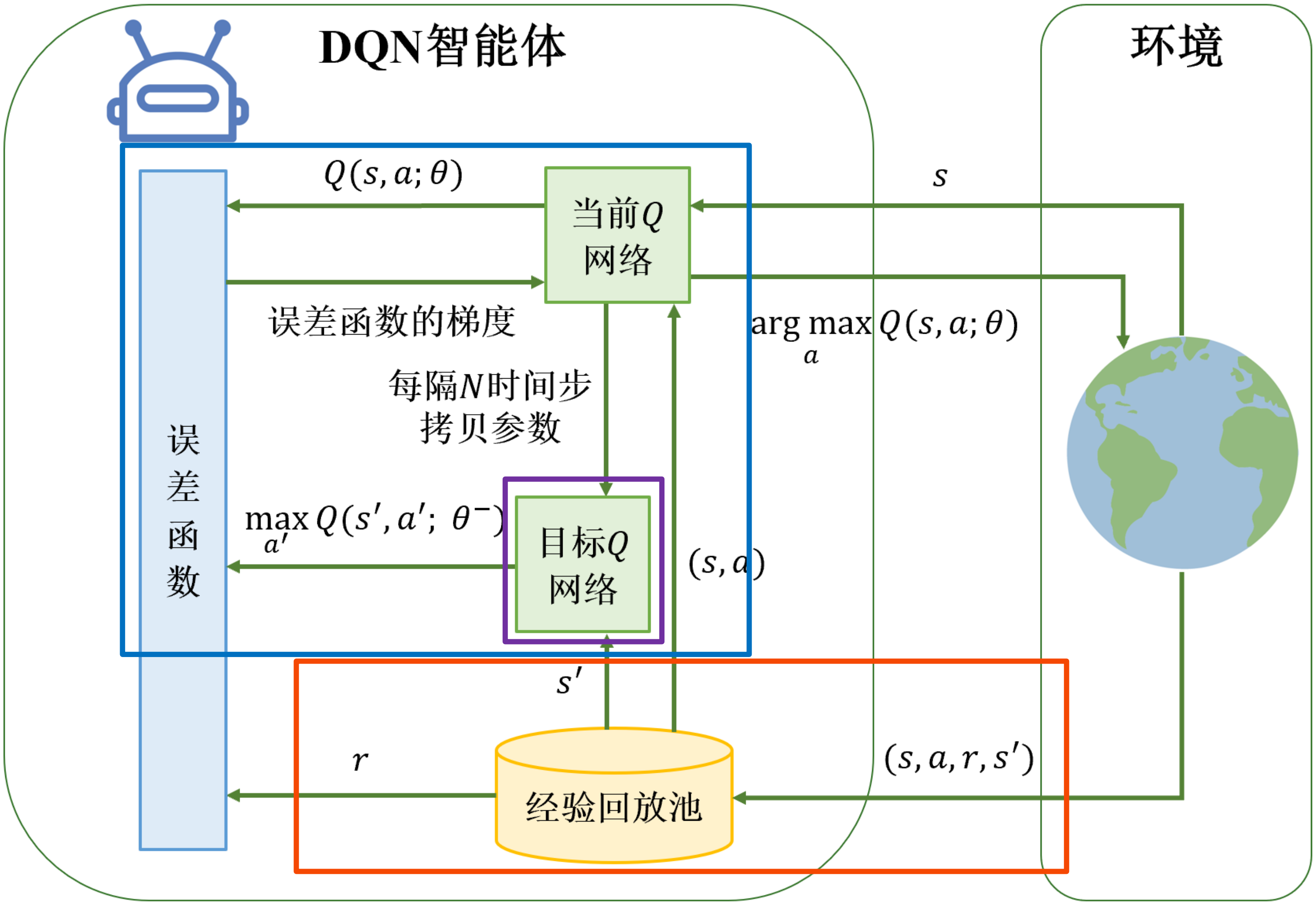
End For

End For

两个网络：目标网络$\hat{Q}$, 评价网络Q

损失函数：$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$

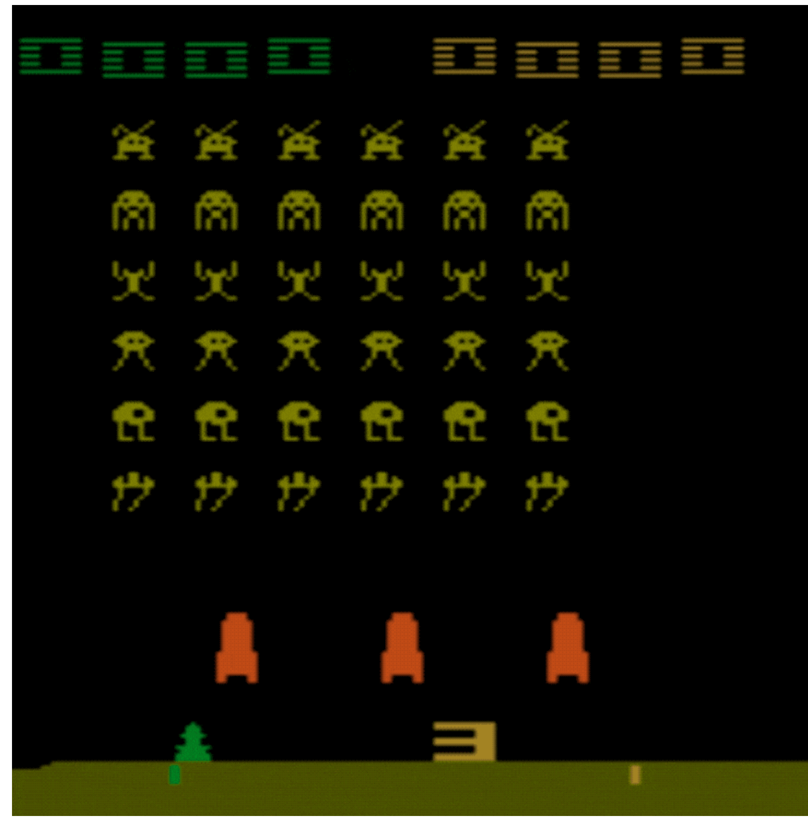
DQN的训练流程


DQN智能体
环境
$Q(s,a;\theta)$
当前Q网络
s
误差函数的梯度
$\arg\max_{a} Q(s,a;\theta)$
每隔N时间步
拷贝参数
误差函数
$\max_{a'} Q(s',a';\theta^-)$
目标Q网络
(s,a)
s'
r
(s,a,r,s')
经验回放池

可视化（续）

Breakout

Space Invaders



DQN在两款游戏中的表现

双重DQN及变种

- DQN的目标Q值

$$Y^{\mathrm{DQN}} = r + \gamma \max_{a'} Q(s', a'; \theta^{-})$$

$$\Updownarrow$$

$$Y^{\mathrm{DQN}} = r + \gamma Q\left(s', \arg\max_{a'} Q(s', a'; \theta^{-}); \theta^{-}\right)$$

- 双重DQN的目标Q值

$$Y^{\mathrm{DoubleDQN}} = r + \gamma Q\left(s', \arg\max_{a'} Q(s', a'; \underline{\theta}); \underline{\theta^{-}}\right)$$

使用参数为θ^-的目标网络来模拟Q值，但使用参数为θ的当前网络来选取最大Q值的动作

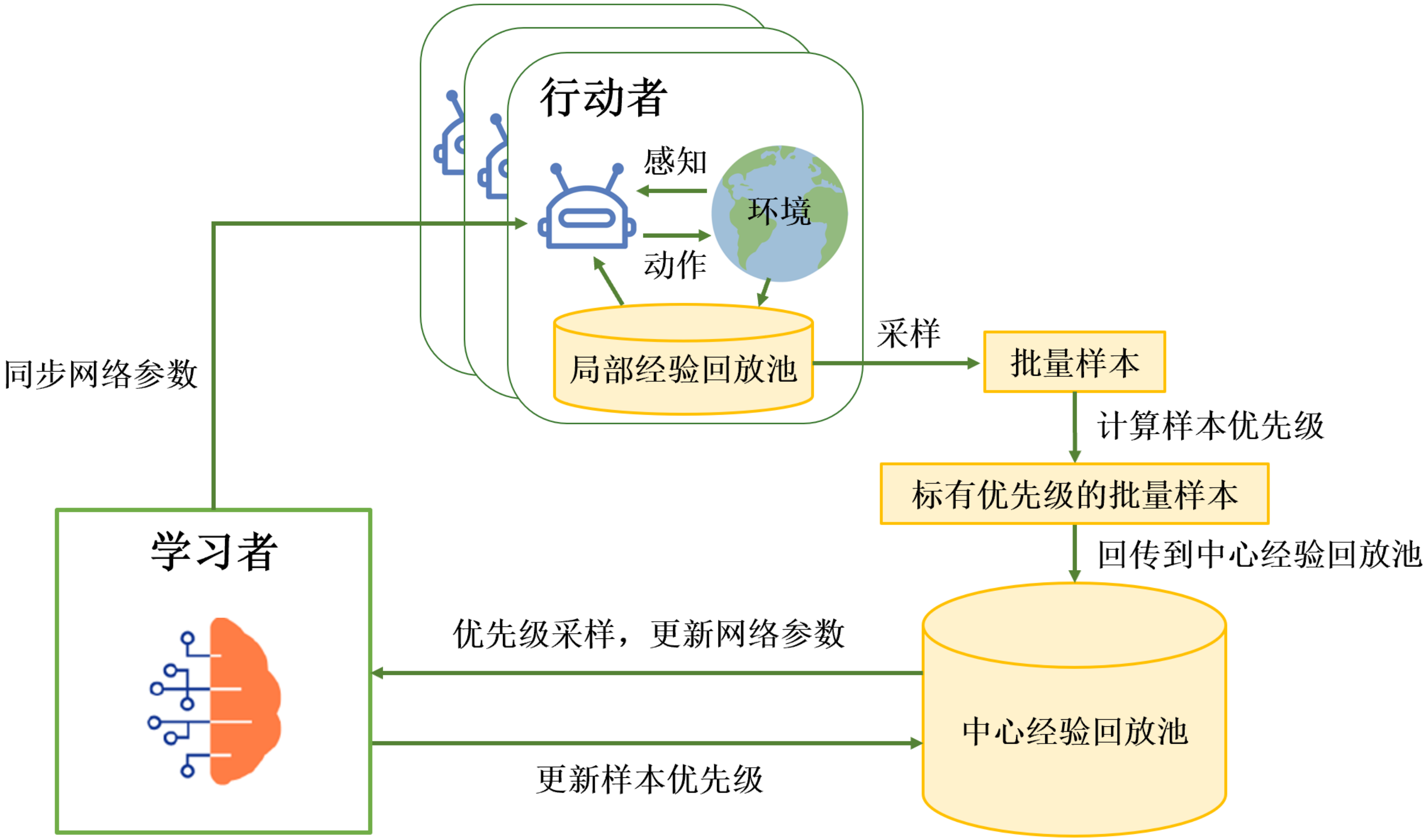
经验回放

- **DQN中的经验回放**
  - 做法：采用均匀采样的方式回放过去的经验
  - 不足：忽略了不同时刻、不同经验的重要程度

- **优先级经验回放**
  - 做法：第i个样本被选择的概率$P(i) = p_i^\alpha / \sum_k p_k^\alpha$
  - 比例优先级：$p_i = |\delta_i| + \epsilon$，其中 δ_i为第i个样本的TD误差
  - 排名优先级：$p_i = \frac{1}{\mathrm{rank}(i)}$，其中rank($i$)为第$i$个样本的优先级排名

- **事后经验回放**
  - 想法：Agent的某条历史轨迹虽然没有完成原定的目标任务，但是可能对另外任务的学习有帮助
  - 可用于多目标、多任务环境下的学习

分布式优先级经验回放

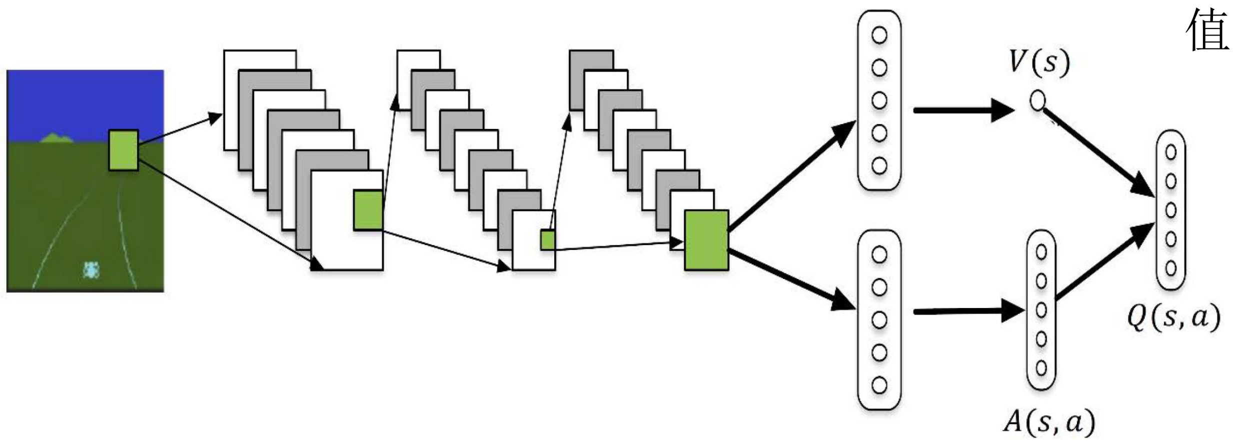


行动者
感知
环境
动作
局部经验回放池
采样
批量样本
计算样本优先级
标有优先级的批量样本
回传到中心经验回放池
同步网络参数
学习者
优先级采样，更新网络参数
中心经验回放池
更新样本优先级

基于竞争（dueling）架构的DQN

$$\boxed{Q(s,a) = V(s) + A(s,a)}$$

值函数　优势函数



- 将Q值分解成V与$A(\boldsymbol{a})$，显式分离出不同动作带来的估计误差

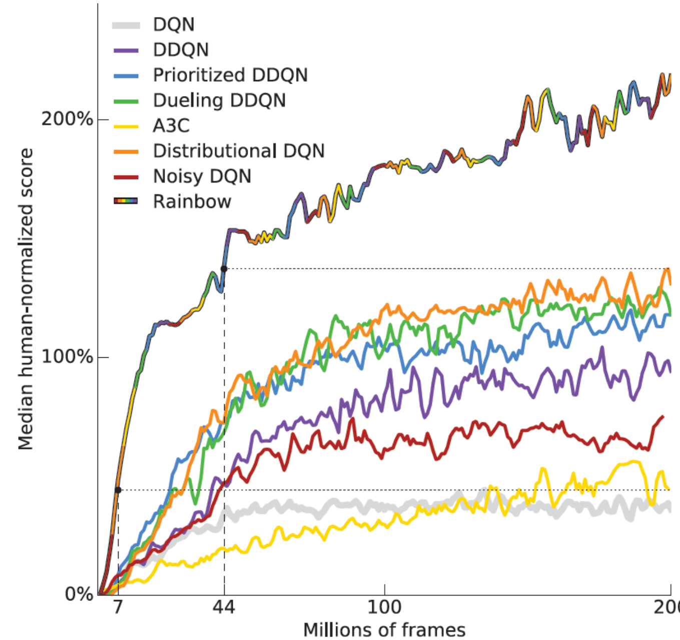
$$Q(s,a_i) = V(s) + \underline{A(s,a_i) - \frac{1}{|A|}\sum_{a_j} A(s,a_j)}$$

对优势函数部分做了中心化的处理

  - V值部分的训练更加稳定，更关注状态部分的信息
  - A值部分更关注细微的动作带来的变化

Rainbow

- 整合DQN中的六种变体
  - 双重DQN
  - 优先级经验回放
  - 竞争架构DQN



- 多步学习
  - 使用n步累计奖励作为目标值

$$\sum_{k=0}^{n-1} \gamma_t^k R(s_{t+k}, a_{t+k}) + \gamma^n \max_{a'} Q(s_{t+n}, a'; \theta^-)$$

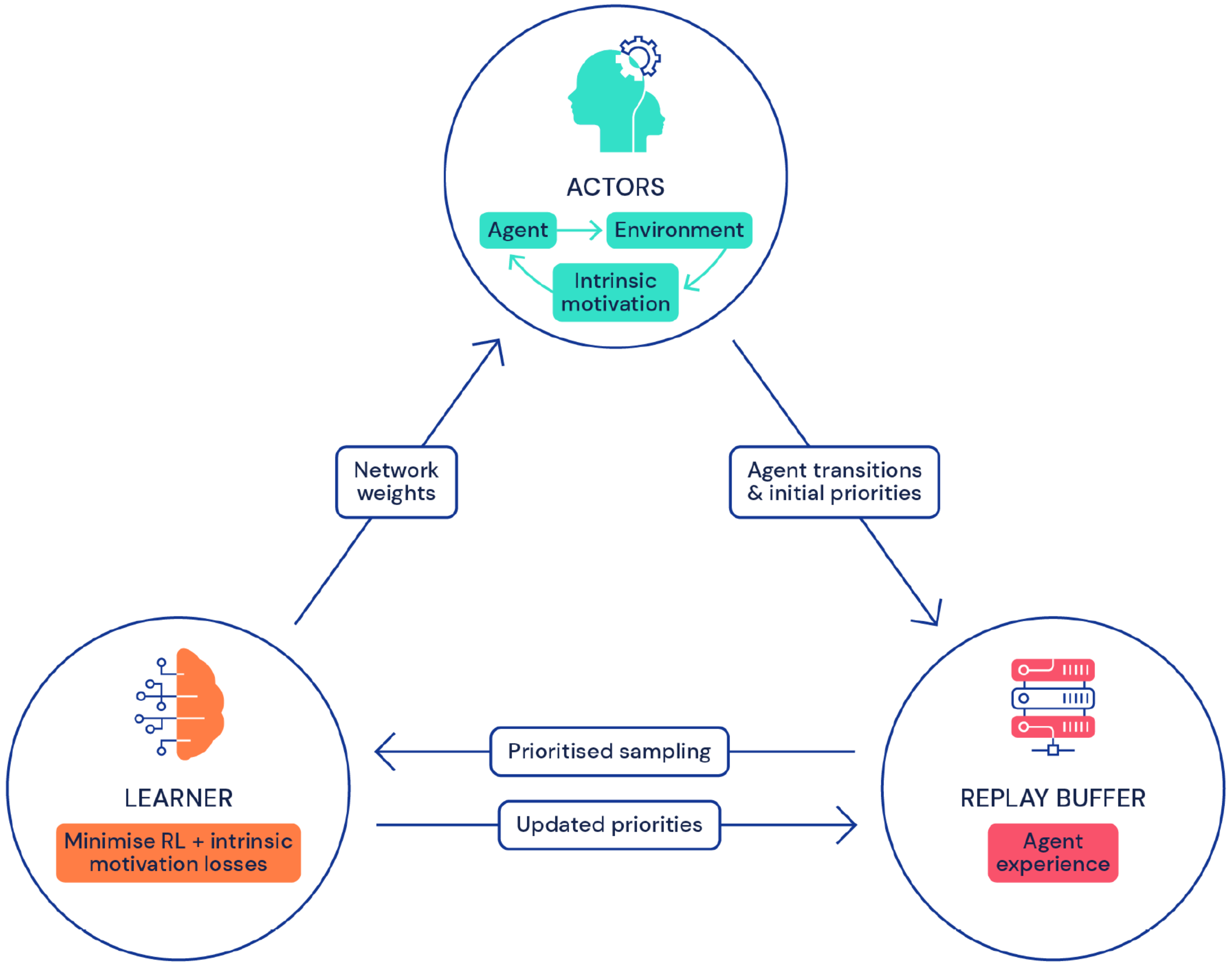
- 基于Q分布的DQN
  - 使用Q值的分布来替代Q值

- 噪声网络
  - 在网络的参数上加噪声，使探索更加系统

2020年

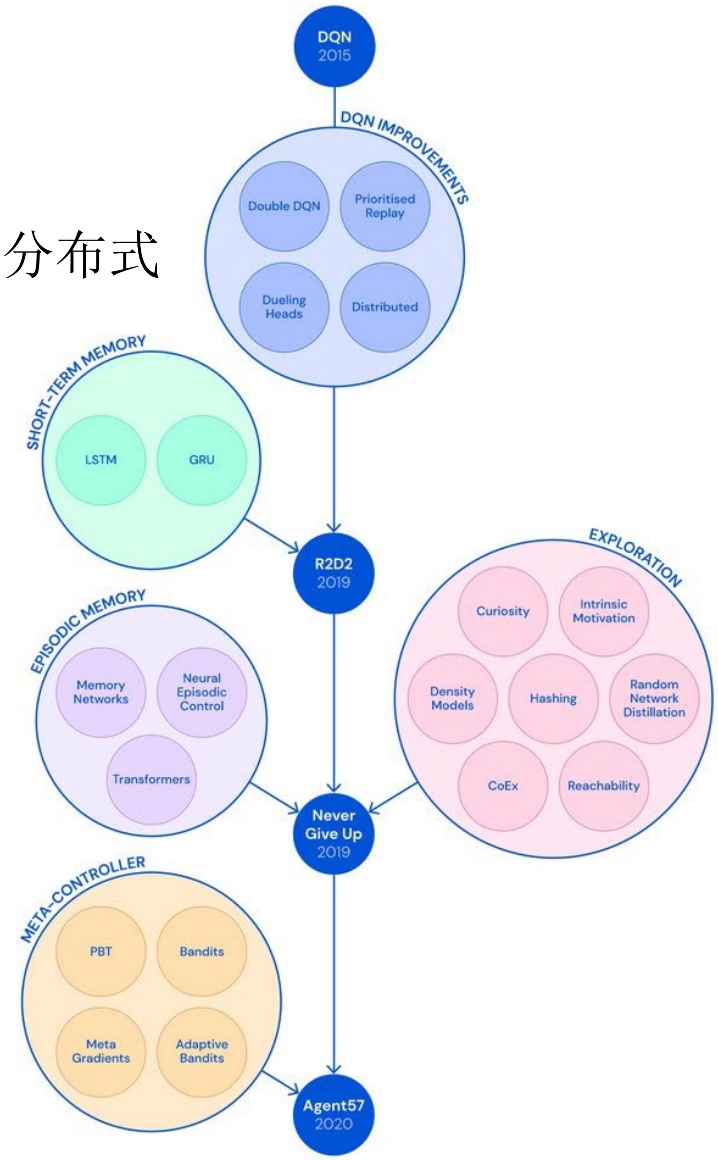

DeepMind @DeepMind · 15h
Our newest blog explores Agent57, the first deep reinforcement learning agent to outperform the human baseline on all 57 Atari games. Read more here:
Agent57: Outperforming the Human Atari Benchmark
Agent57: Outperforming the Human Atari Benchmark
deepmind.com

Agent57总体框架



DQN的改进历程

- 改进的DQN （2015-2018）
  - 双重DQN、优先级经验回放、Dueling、分布式
  - Rainbow：整合DQN中的六种变体
- R2D2（2019）
  - + LSTM、GRU
- Never Give Up (NGU，2019)
  - 情节式记忆
  - 探索
- Agent57（2020）
  - 两个AI模型 + 元控制器



单Agent强化学习

- 强化学习的基本设定
- 动态规划
- 基于值函数的强化学习
- **基于策略梯度的强化学习**

基于值函数 vs. 策略的方法

- 基于值函数的方法：先学习值函数，然后基于值函数选择动作
- 基于策略的方法：学习一个参数化的策略，不需要基于值函数选择动作
  - $\boldsymbol{\theta} \in \mathbb{R}^{d'}$：策略参数向量
  - $\pi(a \mid s, \boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$：给定参数$\boldsymbol{\theta}$，在$t$时刻，在状态$s$采取动作$a$的概率
  - 值函数仍可以被用于学习策略参数，但不用于动作选择
    - 行动者-评论家算法：既学习策略（行动者），又学习值函数（评论家）
    - $\hat{v}(s, \mathbf{w})$：值函数，其中，$\mathbf{w} \in \mathbb{R}^{d}$为参数向量

基于策略梯度的强化学习

① **随机性策略梯度方法**

② 行动者-评论家方法

③ 确定性策略梯度方法

随机性策略梯度方法

- 目标：最大化某性能度量$J(\boldsymbol{\theta})$
- 基于$J(\boldsymbol{\theta})$的梯度来学习$\boldsymbol{\theta}$：

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \underline{\widehat{\nabla J(\boldsymbol{\theta}_t)}}$$

随机梯度上升

$\widehat{\nabla J(\boldsymbol{\theta})} \in \mathbb{R}^{d'}$：梯度$\nabla J(\boldsymbol{\theta})$的随机估计

- 情节式任务： $J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0)$ 在状态s_0执行策略π的真实值
- 随机性策略梯度定理：$\nabla J(\boldsymbol{\theta}) \propto \sum_s \underline{\mu(s)} \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$

$\gamma = 1$的情况

给定策略π时的稳态分布

REINFORCE

REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a)\nabla\pi(a|S_t,\boldsymbol{\theta})\right]$$

$\gamma = 1$的情况

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi\left[\sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a)\frac{\nabla\pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})}\right]$$

$$= \mathbb{E}_\pi\left[q_\pi(S_t,A_t)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}\right]$$

用样本$A_t \sim \pi$替代a

$$= \mathbb{E}_\pi\left[G_t\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}\right]$$

$$\mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t)$$

时刻t的折扣回报 $G_t \doteq \sum_{k=0}^{\infty}\gamma^k R_{t+k}$

- 策略参数向量$\boldsymbol{\theta}$的更新公式：

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

使用基线的REINFORCE

- 不随a变化的任意基线函数$b(s)$满足：

$$\sum_a b(s)\nabla\pi(a|s,\boldsymbol{\theta}) \;=\; b(s)\nabla\sum_a \pi(a|s,\boldsymbol{\theta}) \;=\; b(s)\nabla 1 \;=\; 0$$

- 可以把随机性策略梯度定理推广至包括$b(s)$：

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s)\sum_a \Big(q_\pi(s,a) - b(s)\Big)\nabla\pi(a|s,\boldsymbol{\theta})$$

- 使用基线的REINFORCE的更新公式：

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(G_t - b(S_t)\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

置信域策略优化算法 Trust Region Policy Optimization, TRPO

- 随机性策略梯度方法：基于$J(\boldsymbol{\theta})$的梯度来学习$\boldsymbol{\theta}$：

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \underline{\alpha} \widehat{\nabla J(\boldsymbol{\theta}_t)}$$

如何设置学习率α？

- TRPO：通过把策略的变化限制在置信域内，保证每次策略参数更新能带来策略性能的单调提升

- 存在共轭梯度求解复杂，二阶近似计算量大的问题
  - 改进：近端策略优化算法（Proximal Policy Optimization, PPO）

基于策略梯度的强化学习

① 随机性策略梯度方法

② **行动者-评论家方法**

③ 确定性策略梯度方法

使用基线的REINFORCE vs. 行动者-评论家

- 使用基线的REINFORCE方法：使用了策略和状态值函数，但状态值函数只被用作基线，而不是评论家

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(G_t - \underbrace{b(S_t)}_{\boxed{\hat{v}(S_t,\mathbf{w})}}\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

- 行动者-评论家方法：进一步引入自助（bootstrapping）的思想，以减少方差和加速学习

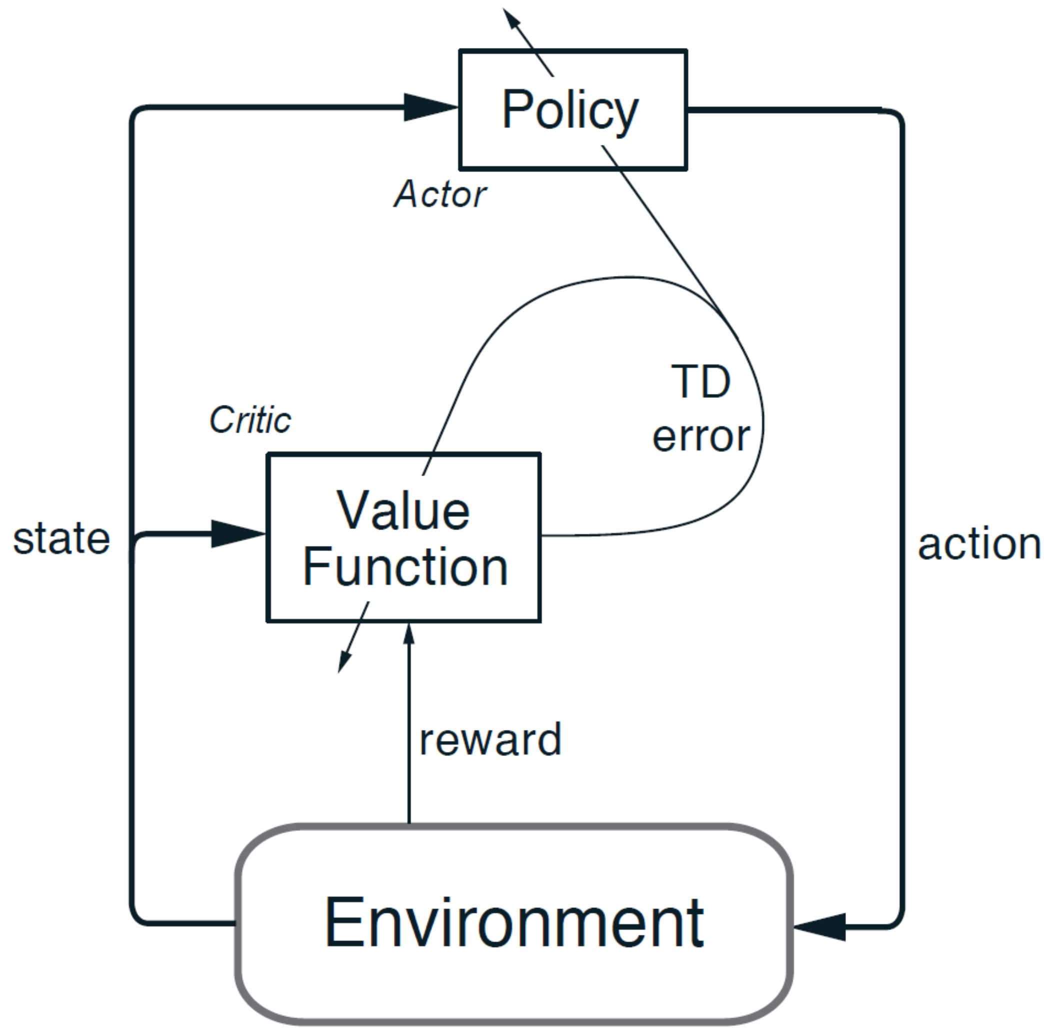
$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\Big(G_{t:t+1} - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha\Big(\underline{R_t + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})}\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha\underline{\delta_t}\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}\end{aligned}$$

1步行动者-评论家：1步TD误差

行动者-评论家架构

主要特点

- 有两个相分离的模块，值函数和策略
  - 值函数：评论家
  - 策略：行动者



- 使用TD误差来同时更新值函数和策略
- 有分离出来的行动者模块的好处：用它来维护一个随机性策略

优势行动者-评论家算法

- 优势函数$A^\pi(s,a)$：在状态s采取动作a的优势

$$A^\pi(s,a) = Q^\pi(s,a) - U^\pi(s)$$

- 优势函数的估计

（1）从$S_t = s, A_t = a$开始，由策略π产生的一条轨迹：

$$S_t, A_t, R_t, S_{t+1}, A_{t+1}, R_{t+1}, S_{t+2}, A_{t+2}, R_{t+2}, \dots$$

（2）使用下式作为优势函数的估计：

$$\underline{\sum_{i=0}^{k-1} \gamma^i R_{t+i} + \gamma^k \hat{v}(s, \mathbf{w})} - \hat{v}(s, \mathbf{w})$$

$\hat{q}(s, a, \mathbf{w})$

- 优势行动者-评论家（A2C）算法
- 异步优势行动者-评论家（A3C）算法

(Asynchronous) Advantage Actor-Critic

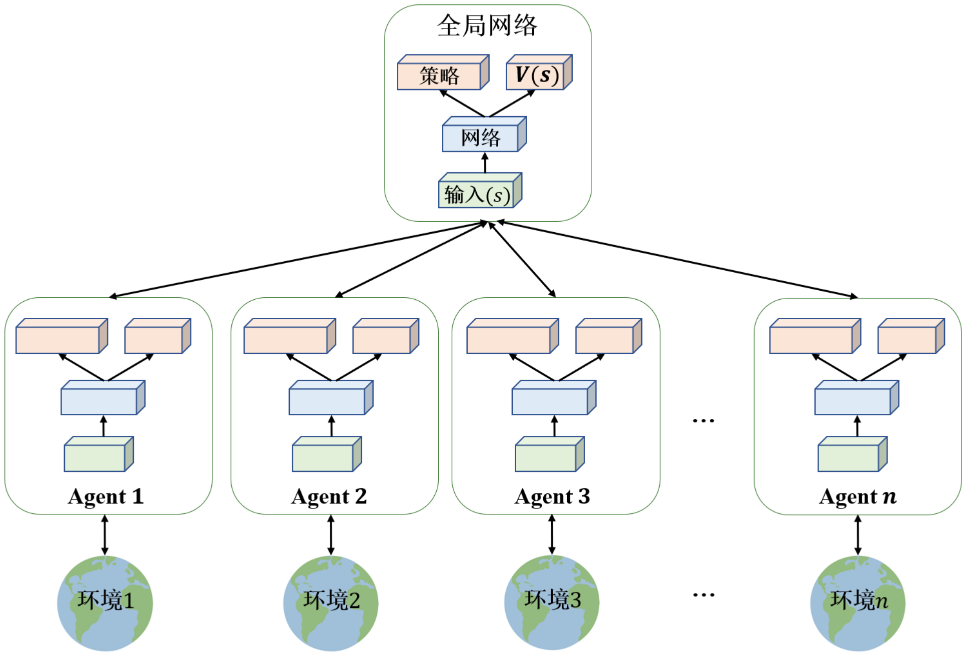
A3C的学习过程

（1）开启多个线程（Agent），从全局网络同步最新的网络参数

（2）每个Agent独立地进行采样、训练学习

（3）每个Agent周期性地独立更新全局网络的参数，即将自己累积的梯度更新到全局网络，然后用最新的全局网络参数对其更新

（4）重复2、3，直到收敛



A3C通过异步学习，发挥多核同时学习的优势

A2C的学习过程

（1）开启多个线程（Agent），从全局网络同步最新的网络参数

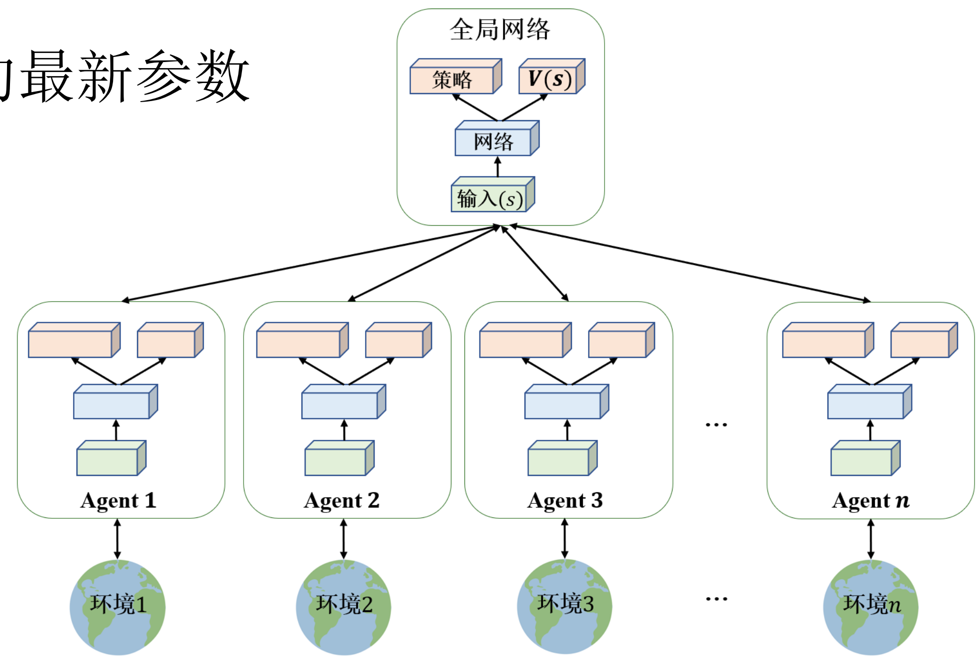
（2）每个Agent独立地进行采样

（3）当数据总量达到mini-batch size时，全部停止采样

（4）全局网络根据mini-batch的数据统一训练学习

（5）每个Agent同步全局网络的最新参数

（6）重复2-5，直到收敛



A3C/A2C通过利用多线程并行独立采样数据

- 保证数据的多样性
- 提高学习效率

柔性行动者-评论家算法

Soft Actor-Critic, SAC

- 目标：找到最优柔性策略

$$\pi^\star = \arg\max_\pi J(\pi) = \arg\max_\pi \sum_{t=0}^{T} \mathbb{E}_\pi \left[\gamma^t (R(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)))\right]$$

  - 温度系数α：平衡环境给出的奖励和策略熵之间的重要程度

柔性值函数 $$V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t)} \left[Q(s_t, a_t) - \alpha \log \pi(a_t \mid s_t)\right]$$

柔性动作值函数 $$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V^\pi(s')$$

- 策略评估 $$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) Q^\pi(s', \pi(s'))$$

- 策略改进 $$\pi_{\text{new}} \leftarrow \arg\min_{\pi \in \Pi} \mathcal{D}_{\text{KL}} \left(\pi(\cdot \mid s_t) \middle\| \frac{\exp\left(\frac{1}{\alpha} Q^{\pi_{\text{old}}}(s_t, \cdot)\right)}{Z^{\pi_{\text{old}}}(s_t)}\right)$$

基于策略梯度的强化学习

① 随机性策略梯度方法

② 行动者-评论家方法

③ **确定性策略梯度方法**

确定性策略梯度算法

Deterministic Policy Gradient, DPG

RL大神：David Silver

- 确定性策略梯度定理：

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s)\ \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right]$$

- 确定性策略：$a = \mu_{\boldsymbol{\theta}}(s)$
  - 优点：需要采样的数据少，算法效率高
  - 缺点：无法探索环境，即给定状态s和策略参数$\boldsymbol{\theta}$时，动作是固定的

- 解决办法：异策略学习，更新过程如下：

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t)$$
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$
$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t)\ \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}$$

深度确定性策略梯度算法

Deep DPG, DDPG

- 解决高维动作空间的控制问题
- 深度：利用深度神经网络逼近动作值函数$Q^w(s,a)$和确定性策略$\mu_{\boldsymbol{\theta}}(s)$
- 使用深度Q网络（DQN）的技巧来更新值网络
  - 经验回放、独立的目标网络
- 扩展算法
  - TD3: 双重延迟（Twin Delayed） DDPG
  - D4PG：分布式分配（Distributed Distributional） DDPG
  - MADDPG：多Agent（Multi-Agent） DDPG

小结

- 强化学习的基本设定
  - 马尔可夫假设、马尔可夫决策过程、策略、值函数
- 基于值函数的强化学习
  - 动态规划：值迭代、策略迭代
  - 表格式的方法：Q学习及变种，深度方法：深度Q网络及变种
- 基于策略梯度的强化学习
  - 随机性能策略梯度方法、行动者-评论家方法、确定性策略梯度方法