深度强化学习笔记

第一章习题：1、A；2、B；3、C；4、B；5、1
第二章习题：1、13.6；2、按一元正态分布采样 $f(x)$ 求均值；
3、构造随机变量 $Z_i = 4f(X_i, Y_i) - \pi$，易知其满足 Bernstein 概率不等式条件，可得

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^{n} 4f(X_i, Y_i) - \pi \geq \epsilon\right|\right) \leq \exp\left(-\frac{\epsilon^2 n/2}{4\pi - \pi^2 + \epsilon\pi/3}\right)$$

令 $\epsilon = \frac{C}{\sqrt{n}}$，则右式为 $\exp\left(-\frac{C^2/2}{v + Cb/(3\sqrt{n})}\right)$，当 $n \to \infty$ 时概率上界为 $\exp\left(-\frac{C^2}{2v}\right)$，则对任意小的概率 δ，选取 $C = \sqrt{-2v\ln\delta}$ 可以使得 $|q_n - \pi|$ 大于 $\frac{C}{\sqrt{n}}$ 的概率不超过 δ，即为原题定理。
4、$n = 1$ 时成立，假设 $n = i - 1$ 时亦成立，

$$q_n = (1 - \frac{1}{n}) \cdot q_{n-1} + \frac{1}{n} \cdot f(x_n) = \frac{n-1}{n} \cdot \frac{1}{n-1}\sum_{i=1}^{n-1} f(x_i) + \frac{1}{n} \cdot f(x_n) = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

第三章习题：1、C；2、D；3、B；4、E
第四章习题：1、B；2、A；3、A；4、789，下；5、18，2；6、B，A；7、A
第五章习题：1、A；2、因为每一步都会利用策略函数 π 采样得到 a 后计算 TD 目标；3、因为更接近无偏估计 u_t
第六章习题：1、A；2、B；3、B；4、B；5、B；6、A；7、C；8、B，C；9、A；10、A；11、ABCDEJ
第七章习题：1、1；2、因为合为一的正数；3、B；4、C；5、随机梯度的近似、动作价值函数的近似；6、A
第八章习题：1、B；2、B
第十一章习题：1、ABD；2、$d_2 * (d_1 + d_2 + 1)$
第十三章习题：1、5；2、A；3、A；4、B
第十五章习题：1、

$$\begin{aligned}\nabla_{\theta^i} J(\theta^1, \cdots, \theta^m) &= \mathbb{E}_{S,A}\Big[\big(Q_\pi(S,A) - b\big) \cdot \nabla_{\theta^i} \ln \pi(A \mid S; \theta^1, \cdots, \theta^m)\Big] \\ &= \mathbb{E}_{S,A}\Big[\big(Q_\pi(S,A) - b\big) \cdot \nabla_{\theta^i} \big(\ln \pi(A^1 \mid S; \theta^1) + \cdots + \ln \pi(A^m \mid S; \theta^m)\big)\Big] \\ &= \mathbb{E}_{S,A}\Big[\big(Q_\pi(S,A) - b\big) \cdot \nabla_{\theta^i} \ln \pi(A^i \mid S; \theta^i)\Big]\end{aligned}$$

第十八章习题：1、A；2、C

基础知识

实际训练神经网络的时候,总是用 SGD（及其变体）而不用 GD。主要原因是 GD 用于非凸问题会陷在鞍点（saddle point），收敛不到局部最优。而 SGD 和小批量（mini-batch）SGD 可

以跳出鞍点，趋近局部最优。另外，GD 每一步的计算量都很大，比 SGD 大 n 倍, 所以 GD 通常很慢(除非用并行计算)。

设 X 是 d 维随机变量，它的取值范围是集合 $\Omega \subset \mathbb{R}^d$。函数 $p(x)$ 是 X 的概率密度函数，设 $f:\Omega \mapsto \mathbb{R}$ 是任意的多元函数，它关于变量 X 的期望是：

$$\mathbb{E}_{X\sim p(\cdot)}[f(X)] = \int_{\Omega} p(x)\cdot f(x)\mathrm{d}x$$

可以在集合 Ω 上均匀采样利用蒙特卡洛计算其定积分，但收敛速度较慢，可以按照概率 $p(x)$ 进行非均匀采样，记作 $x_1,\cdots,x_n\sim p(\cdot)$，并对函数值 $f(x_1),\cdots,f(x_n)$ 求平均 q_n 作为期望 $\mathbb{E}_{X\sim p(\cdot)}[f(X)]$ 的估计值。
如果按照该方式进行采样需要存储 n 个函数值占用内存，可以初始化 $q=0$，从 $t=1$ 到 n，依次计算

$$q_t = \left(1-\frac{1}{t}\right)\cdot q_{t-1} + \frac{1}{t}\cdot f(x_t)$$

易证这样得到的 q_n 等于 $\frac{1}{n}\sum_{i=1}^n f(x_i)$，可以进一步把上式中的 $\frac{1}{t}$ 替换为 α_t，得到公式

$$q_t = (1-\alpha_t)\cdot q_{t-1} + \alpha_t\cdot f(x_t)$$

这个公式叫做 Robbins-Monro 算法，其中 α_n 称为学习步长或学习率，只要 α_t 满足下面的性质，就能保证算法的正确性

$$\lim_{n\to\infty}\sum_{t=1}^n \alpha_t = \infty, \quad \lim_{n\to\infty}\sum_{t=1}^n \alpha_t^2 < \infty$$

我们可以用蒙特卡洛近似期望来理解随机梯度算法，神经网络的训练可以定义为如下优化问题

$$\min_{\omega} \mathbb{E}_{X\sim p(\cdot)}[L(X;\omega)]$$

目标函数 $\mathbb{E}_X[L(X;\omega)]$ 关于 ω 的梯度是（微分算子和积分算子的交换顺序需满足一定条件）

$$g \triangleq \nabla_{\omega}\mathbb{E}_{X\sim p(\cdot)}[L(X;\omega)] = \mathbb{E}_{X\sim p(\cdot)}[\nabla_{\omega}L(X;\omega)]$$

直接计算梯度 g 通常会比较慢，为了加速计算，可以对期望

$$g = \mathbb{E}_{X\sim p(\cdot)}[\nabla_{\omega}L(X;\omega)]$$

做蒙特卡洛近似，把得到的近似梯度 $\tilde{g}$ 称作随机梯度（stochastic gradient），用 $\tilde{g}$ 代替 g 来更新 ω，因为 $\mathbb{E}[\tilde{g}]=g$，它是 g 的一个无偏估计。在实际应用中，样本真实的概率密度函数 $p(x)$ 一般是未知的，在训练神经网络时，我们通常会手机一个训练数据集 $\mathcal{X}=\{x_1,\cdots,x_n\}$，并求解经验风险最小化问题

$$\min_{\omega}\frac{1}{n}\sum_{i=1}^n L(x_i;\omega)$$

这相当于用下面这个概率质量函数代替真实的 $p(x)$

$$p(x)=\begin{cases}\dfrac{1}{n}, & \text{if } x\in\mathcal{X}\\ 0, & \text{if } x\notin\mathcal{X}\end{cases}$$

强化学习的数学基础和建模工具是马尔可夫决策过程（Markov decision process，MDP）。一个 MDP 通常由状态空间 $\mathcal{S}$、动作空间 $\mathcal{A}$、状态转移函数 p、奖励函数 r 和折扣因子 γ 等组成。

书中提到“知道这局游戏的历史记录（即每一步是怎么走的），并不会给你提供额外的信息”，但是对手的风格也是重要信息，alpha go只利用前六步信息，是出于计算量的考虑还是信息和误差的考量？

通常假设奖励是当前状态 s、当前动作 a、下一时刻状态 s' 的函数，把奖励函数记作 $r(s,a,s')$。有时假设奖励仅仅是 s 和 a 的函数，记作 $r(s,a)$。我们总是假设奖励函数是有界的，即对于所有 $a\in\mathcal{A}$ 和 $s,s'\in s$，有 $|r(s,a,s')|<\infty$。

状态转移概率函数（state transition probability function），通常假设为平稳的，即不随着时刻 t 变化

$$p_t(s'\mid s,a)=\mathbb{P}(S'_{t+1}=s'\mid S_t=s,A_t=a)$$

轨迹（trajectory）是指一回合（episode）游戏中，智能体观测到的所有的状态、动作、奖励：

$$s_1,a_1,r_1,\quad s_2,a_2,r_2,\quad s_3,a_3,r_3,\cdots$$

回报（return）是从当前时刻开始到本回合结束的所有奖励的总和，所以回报也叫做累计奖励（cumulative future reward）。使用折扣回报（discounted return）作为折扣后的总奖励：

$$U_t=R_t+\gamma\cdot R_{t+1}+\gamma^2\cdot R_{t+2}+\gamma^3\cdot R_{t+3}+\cdots$$

MDP 的时间步可以是有限期（finite-horizon）或无限期（infinite-horizon）。有限期 MDP 存在一个终止状态（terminal state），该状态被智能体触发后，一个回合（episode）结束。与之对应的是无限期 MDP，即环境中不存在终止状态，在折扣率 $\gamma\geq 1$ 时会导致回报趋于无穷。

假设对于所有的 $s\in\mathcal{S}$ 和 $a\in\mathcal{A}$，回报函数有界 $|r(s,a)|<b$，那么对于 $\gamma\in[0,1)$，且 $r(s,a)$，有这样的性质：

$$\left|\lim_{n\to\infty}\sum_{i=t}^{n}\gamma^{i-t}r_i\right|\leq\frac{b}{1-\gamma}$$

假设观测到状态 s_t，选中动作 a_t，则回报 U_t 关于随机变量求条件期望，得到

$$Q_\pi(s_t,a_t)=\mathbb{E}_{S_{t+1},A_{t+1},\cdots,S_n,A_n}[U_t\mid S_t=s_t,A_t=a_t]$$

条件期望的结果 $Q_\pi(s_t,a_t)$ 被称作动作价值函数（action-value function），更准确地说应该是动作状态价值函数。

为了排除掉策略 π 的影响，只评价当前状态和动作的好坏，可以利用最优动作价值函数（optimal action-value function）：

$$Q_*(s_t, a_t) = \max_{\pi} Q_\pi(s_t, a_t), \quad \forall s_t \in \mathcal{S}, a_t \in \mathcal{A}$$

如果只想知道当前状态 s_t 是否对自己有利，可以利用状态价值函数（state-value function）：

$$V_\pi(s_t) = \mathbb{E}_{A_t \sim \pi(\cdot|s_t)}[Q_\pi(s_t, A_t)] = \sum_{a \in \mathcal{A}} \pi(a \mid s_t) \cdot Q_\pi(s_t, a)$$

状态价值函数 $V_\pi(s_t)$ 也是回报 U_t 的期望：

$$V_\pi(s_t) = \mathbb{E}_{A_t, S_{t+1}, A_{t+1}, \cdots, S_n, A_n}[U_t \mid S_t = s_t]$$

价值学习

DQN 与 Q 学习

最优动作价值函数可以用来做控制，一旦知道状态 s_t，可以根据对动作的打分进行选择。

近似学习 Q_* 最有效的方法是深度 Q 网络（deep Q network，DQN），记作 $Q(s, a; \omega)$。训练 DQN 最常用的算法是时间差分（temporal difference，TD）。

假如在出发前，模型预估北京到上海需要 $\hat{q} = 14$ 小时，经过 $r = 4.5$ 小时从北京到达济南后，再让模型预测从济南到上海需要 $\hat{q}' = 11$ 个小时，则整个旅程的最新估计时间为

$$\hat{y} \triangleq r + \hat{q}' = 4.5 + 11 = 15.5$$

TD 算法将 $\hat{y} = 15.5$ 称为 TD 目标（TD target），它比最初的预测 $\hat{q} = 14$ 更可靠，因为它是纯粹估计的，而 TD 目标中含有事实的成分。我们希望估计值 $\hat{q}$ 尽量接近 TD 目标 $\hat{y}$，所以用两者差的平方作为损失函数：

$$L(\omega) = \frac{1}{2}\big[Q(\text{〃北京〃}, \text{〃上海〃}; \omega) - \hat{y}\big]^2$$

此处把 $\hat{y}$ 看做常数，尽管它依赖于 ω。计算损失函数的梯度：

$$\nabla_\omega L(\omega) = \underbrace{(\hat{q} - \hat{y})}_{\delta} \cdot \nabla_\omega Q(\text{〃北京〃}, \text{〃上海〃}; \omega)$$

此处的 $\delta = \hat{q} - \hat{y} = 14 - 15.5 = -1.5$ 称作 TD 误差（TD error）。

由下列两式推导

$$U_t = R_t + \gamma \cdot \underbrace{\sum_{k=t+1}^{n} \gamma^{k-t-1} \cdot R_k}_{=U_{t+1}}, \qquad Q_*(s_t, a_t) = \max_{\pi} \mathbb{E}[U_t \mid S_t = s_t, A_t = a_t]$$

可得下列定理，该定理是最优贝尔曼方程（optimal Bellman equations）的一种形式

$$\underbrace{Q_*(s_t,a_t)}_{U_t\text{的期望}}=\mathbb{E}_{S_{t+1}\sim p(\cdot|s_t,a_t)}[R_t+\gamma\cdot\underbrace{\max_{A\in\mathcal{A}}Q_*(S_{t+1},A)}_{U_{t+1}\text{的期望}}\mid S_t=s_t,A_t=a_t]$$

贝尔曼方程的右边是个期望，我们可以对期望做蒙特卡洛近似，当智能体执行动作 a_t 后，环境通过状态转移函数 $p(s_{t+1}\mid s_t,a_t)$ 计算出新状态 s_{t+1}，那么当我们观测到 s_t, a_t, s_{t+1} 后则奖励 r_t 也被观测到，因此可以计算

$$r_t+\gamma\cdot\max_{a\in\mathcal{A}}Q_*(s_{t+1},a)$$

它可以看做是右式期望的近似。因此，如果用神经网络 $Q(s,a;\omega)$ 替换最优动作价值函数 $Q_*(s,a)$ 可以得到

$$\underbrace{Q(s_t,a_t;\omega)}_{\text{预测}\hat{q}_t}\approx\underbrace{r_t+\gamma\cdot\max_{a\in\mathcal{A}}Q(s_{t+1},a;\omega)}_{\text{TD 目标}\hat{y}_t}$$

应当鼓励 $\hat{q}_t\triangleq Q(s_t,a_t;\omega)$ 接近 $\hat{y}_t$，定义损失函数

$$L(\omega)=\frac{1}{2}[Q(s_t,a_t;\omega)-\hat{y}_t]^2$$

假设 $\hat{y}$ 是常数，计算 L 关于 ω 的梯度

$$\nabla_\omega L(\omega)=\underbrace{(\hat{q}_t-\hat{y}_t)}_{\text{TD 误差}\delta_t}\cdot\nabla_\omega Q(s_t,a_t;\omega)$$

做一步梯度下降，可以让 $\hat{q}_t$ 更接近 $\hat{y}_t$

$$\omega\leftarrow\omega-\alpha\cdot\delta_t\cdot\nabla_\omega Q(s_t,a_t;\omega)$$

收集训练数据：我们可以用任何策略函数 π 去控制智能体与环境交互，这个 π 就叫做行为策略（behavior policy）。比较常用的是 ϵ-greedy 策略（初始时将 ϵ 设置得比较大，然后逐渐衰减）：

$$a_t=\begin{cases}\arg\max_a Q(s_t,a;\omega), & \text{以概率 } (1-\epsilon);\\ \text{均匀抽取 } \mathcal{A} \text{ 中的一个动作}, & \text{以概率 } \epsilon.\end{cases}$$

把智能体在一局游戏中的轨迹记作 $s_1,a_1,r_1,\quad,s_2,a_2,r_2,\cdots,\quad s_n,a_n,r_n$。把一条轨迹划分成 n 个 (s_t,a_t,r_t,s_{t+1}) 这种四元组，存入数组，这个数组叫做经验回放数组（replay buffer）。

更新 DQN 参数 ω：随机从经验回放数组中取出一个四元组，记作 (s_j,a_j,r_j,s_{j+1})，执行下列步骤进行参数更新：

1. 对 DQN 做正向传播，得到 Q 值：

$$\hat{q}_j=Q(s_j,a_j;\omega_{\text{now}})\text{ 和 }\hat{q}_{j+1}=\max_{a\in\mathcal{A}}Q(s_{j+1},a;\omega_{\text{now}})$$

2. 计算 TD 目标和 TD 误差：

$$\hat{y}_j = r_j + \gamma \cdot \hat{q}_{j+1} \text{ 和 } \delta_j = \hat{q}_j - \hat{y}_j$$

3. 对 DQN 作反向传播，得到梯度：

$$g_j = \nabla_{\omega} Q(s_j, a_j; \omega_{\text{now}})$$

4. 做梯度下降更新 DQN 的参数：

$$\omega_{\text{now}} \leftarrow \omega_{\text{now}} - \alpha \cdot \delta_j \cdot g_j$$

   可以在智能体每执行一个动作之后，对 ω 做几次更新，也可以在每完成一局游戏后，对 ω 做几次更新。

利用表格法实现 Q 学习，首先用表格表示 Q_*，即各状态下各动作的最优动作价值函数，用下列公式更新表格中元素

$$\hat{y}_t \triangleq r_t + \gamma \cdot \max_{a \in \mathcal{A}} \tilde{Q}(s_{t+1}, a); \qquad \tilde{Q}(s_t, a_t) \leftarrow (1-\alpha) \cdot \tilde{Q}(s_t, a_t) + \alpha \cdot \hat{y}_t$$

行为策略（behavior policy）的作用是收集经验（experience），即观测的状态、动作、奖励；目标策略是用来控制智能体的策略函数。同策略（on-policy）是指利用相同的行为策略和目标策略，异策略（off-policy）是指用不同的行为策略和目标策略。

异策略的好处是可以用行为策略收集经验，将 (s_t, a_t, r_t, s_{t+1}) 存储作为经验回放数组，利用这些数据训练目标策略被称作经验回放（experience relpay）。

SARSA 算法

传统的强化学习用 Q_π 作为确定性的策略控制智能体，但是现在 Q_π 通常被用于评价策略的好坏，而非用于控制智能体。Q_π 常与策略函数 π 结合使用，被称作 actor-critic（演员-评委）方法。

SARSA 算法的表格法与 Q 学习类似，利用下列贝尔曼方程

$$Q_\pi(s_t, a_t) = \mathbb{E}_{S_{t+1}, A_{t+1}}[R_t + \gamma \cdot Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s_t, A_t = a_t]$$

其中左式可近似为表格中的 $q(s_t, a_t)$，右式可根据 s_{t+1} 和策略 π 做随机抽样，得到新的动作

$$\tilde{a}_{t+1} \sim \pi(\cdot \mid s_{t+1})$$

并用观测到的 r_t、s_{t+1} 和计算出的 $\tilde{a}_{t+1}$ 对期望做蒙特卡洛近似，然后用表格 q 近似 Q_π 得到

$$\hat{y}_t \triangleq r_t + \gamma \cdot q(s_{t+1}, \tilde{a}_{t+1})$$

并用下式更新表格中 (s_t, a_t) 位置上的元素

$$q(s_t, a_t) \leftarrow (1-\alpha) \cdot q(s_t, a_t) + \alpha \cdot \hat{y}_t$$

然后用某种算法更新策略函数，注意这是隐性的更新，通过更新 Q 函数来间接改进策略。

SARSA 算法的目标是学到表格 q 作为动作价值函数 Q_π 的近似，经验回放数组里的经验 (s_j, a_j, r_j, s_{j+1}) 是过时的行为策略 π_{old} 收集到的，与当前策略 π_{now} 及其对应的价值 $Q_{\pi_{\text{now}}}$ 对应不上，因此不能使用经验回放，只能使用同策略。

可以用神经网络 $q(s, a; \omega)$ 来近似 $Q_\pi(s, a)$，称其为价值网络（value network），其训练流程基本与训练 DQN 完全相同，只是需要根据当前策略做抽样

$$\tilde{a}_{t+1} \sim \pi_{\text{now}}(\cdot \mid s_{t+1})$$

SARSA 计算 TD 目标只用到一个奖励 r_t，这样得到的 $\hat{y}_t$ 叫做单步 TD 目标，可将其扩展为多步 TD 目标。易知

$$U_t = \left(\sum_{i=0}^{m-1} \gamma^i R_{t+i} \right) + \gamma^m U_{t+m}$$

动作价值函数 $Q_\pi(s_t, a_t)$ 是回报 U_t 的期望，而 $Q_\pi(s_{t+m}, a_{t+m})$ 是回报 U_{t+m} 的期望，则可以得到下列定理：
设 R_k 是 S_k、A_k、S_{k+1} 的函数，$\forall k = 1, \cdots, n$，则

$$\underbrace{Q_\pi(s_t, a_t)}_{U_t\text{的期望}} = \mathbb{E}\left[\left(\sum_{i=0}^{m-1} \gamma^i R_{t+i} \right) + \gamma^m \cdot \underbrace{Q_\pi(S_{t+m}, A_{t+m})}_{U_{t+m}\text{的期望}} \middle| S_t = s_t, A_t = a_t \right]$$

所以已知当前状态 s_t，用策略 π 控制智能体与环境交互 m 次，得到轨迹（不需要 r_{t+m}）

$$r_t, \quad s_{t+1}, a_{t+1}, r_{t+1}, \quad \cdots, \quad s_{t+m-1}, a_{t+m-1}, r_{t+m-1}, \quad s_{t+m}, a_{t+m}$$

后可以计算 m 步 TD 目标用以更新价值网络和策略

$$\hat{y}_t = \left(\sum_{i=0}^{m-1} \gamma^i r_{t+i} \right) + \gamma^m \cdot q(s_{t+m}, a_{t+m}; \omega)$$

训练价值网络 $q(s, a; \omega)$ 时，可以将一局游戏进行到底，计算回报 $u_t = \sum_{i=0}^{n-t} \gamma^i r_{t+i}$，使用 u_t 作为目标鼓励价值网络接近 u_t 不是 TD 方法，而是蒙特卡洛，因为这是用实际观测 u_t 去近似期望。

- 蒙特卡洛的好处是无偏性：u_t 是 $Q_\pi(s_t, a_t)$ 的无偏估计，由于 u_t 的无偏性，拿其作为目标训练价值网络，得到的价值网络也是无偏的。
- 蒙特卡洛的坏处是方差大：随机变量 U_t 依赖于 S_{t+1}，A_{t+1}，$\cdots$，S_n，A_n 这些随机变量，其中不确定性很大，可能导致观测值 u_t 实际上离 $\mathbb{E}[U_t]$ 很远，因此拿 u_t 作为目标训练价值网络收敛会很慢。

SARSA 算法中用价值网络自己做出的估计去更新价值网络自己，这属于自举（bootstrapping）。

- 自举的好处是方差小。单步 TD 目标的随机性只来自于 S_{t+1} 和 A_{t+1}，而回报 U_t 的随机性来自于 S_{t+1}，A_{t+1}，$\cdots$，S_n，A_n。很显然单步 TD 目标的随机性较小，因此方差较小，

用自举训练价值网络，收敛比较快。

- 自举的坏处是有偏差。价值网络 $q(s,a;\omega)$ 是对动作价值 $Q_\pi(s,a)$ 的近似，假如 $q(s_{j+1},a_{j+1};\omega)$ 低估（或高估）真实价值 $Q_\pi(s_{j+1},a_{j+1})$，则会导致 $q(s_j,a_j;\omega)$ 低估（高估）$Q_\pi(s_j,a_j)$。也就是说，自举会让偏差从 (s_{t+1},a_{t+1}) 传播到 (s_t,a_t)。

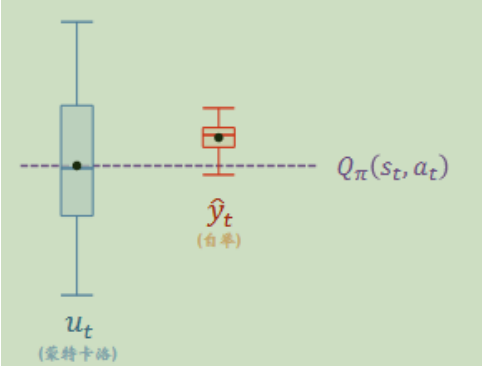

图 5.5: u_t 和 $\hat{y}_t$ 的箱型图（boxplot）示意。

价值学习高级技巧

把智能体的轨迹划分成 s_t,a_t,r_t,s_{t+1} 这样的四元组，存入数组作为经验回放数组，需要认为制定超参数数组的大小（记作 b），数组中只保留最近 b 条数据。实践中要等回放数组有最够多的四元组时才开始更新 DQN。

经验回放的一个好处在于打破序列的相关性，训练 DQN 的相邻四元组应该是独立的，然而当智能体收集经验的时候，相邻两个四元组 (s_t,a_t,r_t,s_{t+1}) 和 $(s_{t+1},a_{t+1},r_{t+1},s_{t+1})$ 有很强的相关性，如果是依次使用这些强关联的四元组训练 DQN 效果会很差，因此通常随机抽选四元组更新 DQN。

经验回放局限性是只能适用于异策略，策略的变化导致收集经验时用的行为策略是过时的策略。

优先经验回放（prioritized experience replay）会给每个四元组一个权重，然后根据权重做非均匀随机抽样。如果 DQN 对 (s_j,a_j) 的价值判断不准确，即 $Q(s_j,a_j;\omega)$ 离 $Q_*(s_j,a_j)$ 较远，则四元组 (s_j,a_J,r_j,s_{j+1}) 应当有较高的权重。

利用 TD 误差代替无法观测的 $|Q(s_j,a_j;\omega)-Q_*(s_j,a_j)|$，如果 TD 误差的绝对值 $|\delta_j|$ 大，说明 DQN 对 (s_j,a_j) 的真实价值的评估不准确，应该给 (s_j,a_j,r_j,s_{j+1}) 设置较高的权重。

有两种方法设置抽样概率，分别是

$$p_j \propto |\delta_j|+\epsilon, \qquad p_j \propto \frac{1}{\mathrm{rank}(j)}$$

优先经验回放做非均匀抽样，这会导致 DQN 的预测有偏差，应当相应调整学习率，抵消掉不同抽样概率造成的偏差。如果一条样本被抽样的概率大，那么它的学习率就应该小，可以这样设置学习率：

$$\alpha_j = \frac{\alpha}{(b\cdot p_j)^\beta}$$

此处的 b 是经验回放数组中样本的总数，$\beta \in (0,1)$ 是个需要调的超参数（论文中建议一开始让 β 较小，逐渐增长到 1）。

当 $\beta = 1$ 时按照上述学习率的调整，抽样概率和学习率应该被完全抵消，但实际上设置学习率为 $\frac{\alpha}{10}$，使用样本 (s_j, a_j, r_j, s_{j+1}) 计算十次梯度，更新十次参数 ω 是对样本更有效的利用，第二种方式的缺点在于计算量增加了。（如何证明？以及按照这种思想岂不是所有基于损失函数梯度的优化算法都能进行改进？）

高估问题：用 Q 学习训练出的 DQN 会高估真实的价值，并且高估通常是非均匀的（什么意思？整个动作-状态空间的中高估不是恒定的)。

1. 自举导致偏差的传播：如果 $Q(s_{j+1}, a_{j+1}; \omega)$ 是对真实价值 $Q_*(s_{j+1}, a_{j+1})$ 的低估（或高估），就会导致 $Q(s_j, a_j; \omega)$ 低估（或高估）价值 $Q_*(s_j, a_j)$。
2. 即使 DQN 是真实价值 Q_* 的无偏估计，只要 DQN 不恒等于 Q_*，TD 目标就会高估真实价值。TD 目标是高估，而 Q 学习算法鼓励 DQN 预测接近 TD 目标，因此 DQN 会出现高估。

往任意 d 个实数 $x_1, \cdots, x_d$ 中加入任意均值为零的随机噪声，得到随机变量 $Z_1, \cdots, Z_d$。易证均值为零的随机噪声不会影响均值：

$$\mathbb{E}\left[\text{mean}(Z_1, \cdots, Z_d)\right] = \text{mean}(x_1, \cdots, x_d)$$

用稍微复杂一点的证明，可以得到

$$\mathbb{E}\left[\max(Z_1, \cdots, Z_d)\right] \geq \max(x_1, \cdots, x_d)$$

假设对于所有的动作 $a \in \mathcal{A}$ 和状态 $s \in \mathcal{S}$，DQN 的输出 $Q(s, a; \omega)$ 是真实价值 $Q_*(s, a)$ 加上均值为零的随机噪声 ϵ，显然 $Q(s, a; \omega)$ 是对真实价值 $Q_*(s, a)$ 的无偏估计，但是

$$\mathbb{E}_\epsilon \left[\max_{a \in \mathcal{A}} Q(s, a; \omega)\right] \geq \max_{a \in \mathcal{A}} Q_*(s, a)$$

根据 TD 目标

$$\hat{y}_j = r_j + \gamma \cdot \underbrace{\max_{a \in \mathcal{A}} Q(s_{j+1}, a; \omega)}_{\text{高估 } \max_{a \in \mathcal{A}} Q_*(s_{j+1}, a)}$$

所以 TD 目标通常是对真实价值 $Q_*(s_j, a_j)$ 的高估，TD 算法鼓励 $Q(s_j, a_j; \omega)$ 接近 TD 目标，这将导致 $Q(s_j, a_j; \omega)$ 高估真实价值 $Q_*(s_j, a_j)$。

切断自举可以避免误差的传播，从而缓解 DQN 的高估，可以用另一个神经网络计算 TD 目标，而不是用 DQN 自己计算 TD 目标。另一个神经网络被称作目标网络（target network），记作 $Q(s, a; \omega^-)$，它的神经网络结构与 DQN 完全相同，但是参数 ω^- 不同于 ω。其参数更新步骤如下：

1. 对 DQN 做正向传播，得到

$$\hat{q}_j = Q(s_j, a_j; \omega_{\text{now}})$$

2. 对目标网络做正向传播，得到

$$\hat{q}_{j+1}^- = Q(s_j, a_j; \omega_{\text{now}}^-)$$

3. 计算 TD 目标和 TD 误差

$$\hat{y}_j^- = r_j + \gamma \cdot \hat{q}_{j+1}^-, \qquad \delta_j = \hat{q}_j - \hat{y}_j^-$$

4. 对 DQN 做反向传播得到梯度 $\nabla_\omega Q(s_j, a_j; \omega_{\text{now}})$
5. 做梯度下降更新 DQN 的参数

$$\omega_{\text{new}} \leftarrow \omega_{\text{now}} - \alpha \cdot \delta_j \nabla_\omega Q(s_j, a_j; \omega_{\text{now}})$$

6. 设 $\tau \in (0, 1)$ 是需要手动调的超参数，做加权平均更新目标网络的参数（τ 通常是一个非常小的数，例如0.001、0.005）

$$\omega_{\text{new}}^- \leftarrow \tau \cdot \omega_{\text{new}} + (1 - \tau) \cdot \omega_{\text{now}}^-$$

   为什么目标网络要用 DQN 的参数来更新？这个公式的本质是让目标网络的参数缓慢地、平滑地追踪主网络的参数，在“稳定”与“及时”之间取得平衡。

改用目标网络计算 $\hat{y}$ 避免了用 DQN 的估计更新 DQN 自己，降低自举的危害，然而这种方法不能完全避免自举，原因是目标网络的参数仍然与 DQN 相关。

双 Q 学习：下一步最佳动作的选择依赖 DQN

$$a^* = \arg\max_{a \in \mathcal{A}} Q(s_{j+1}, a; \omega)$$

其 TD 目标的求值使用目标网络

$$\tilde{y}_j = r_j + Q(s_{j+1}, a^*; \omega^-)$$

因为

$$\underbrace{Q(s_{j+1}, a^*; \omega^-)}_{\text{双 Q 学习}} \leq \underbrace{\max_{a \in \mathcal{A}} Q(s_{j+1}, a; \omega^-)}_{\text{用目标网络的 Q 学习}}$$

所以 $\underbrace{\tilde{y}_t}_{\text{双 Q 学习}} \leq \underbrace{\max_{a \in \mathcal{A}} Q(s_{j+1}, a; \omega^-)}_{\text{用目标网络的 Q 学习}}$，说明用双 Q 学习得到的 TD 目标更小，较目标网络的 Q 学习缓解了高估。

在实践中应当尽量使用双 Q 学习，它是原始 Q 学习、Q 学习+目标网络、双 Q 学习中效果最好的。如果使用 SARSA 算法，自举的问题依然存在，但是不存在最大化造成高估这一问题，因此只需要应用目标网络即可。

对决网络（dueling network）基本想法是将最优动作价值 Q_* 分解成最优状态价值 V_* 加最优优势 D_*，其训练与 DQN 完全相同，可以用 Q 学习算法或者双 Q 学习算法。

状态价值函数 $V_\pi(s)$ 是 $Q_\pi(s,a)$ 关于 a 的期望 $V_\pi(s)=\mathbb{E}_{A\sim\pi}[Q_\pi(s,A)]$，最优状态价值函数 V_* 的定义是

$$V_*(s)=\max_\pi V_\pi(s),\quad \forall s\in\mathcal{S}$$

最优优势函数（optimal advantage function）$D_*(s,a)$ 的定义是

$$D_*(s,a)\triangleq Q_*(s,a)-V_*(s)$$

通过数学推导，可以证明下面定理

$$Q_*(s,a)=V_*(s)+D_*(s,a)-\underbrace{\max_{a\in\mathcal{A}}D_*(s,a)}_{\text{恒等于零}},\qquad \forall s\in\mathcal{S},\forall a\in\mathcal{A}$$

对决网络由近似最优优势函数 $D_*(s,a)$ 的神经网络 $D(s,a;\omega^D)$ 和近似最优状态价值函数 $V_*(s)$ 的神经网络 $V(s;\omega^V)$ 组成，则最优动作价值函数 Q_* 就被近似为

$$Q(s,a;\omega)\triangleq V(s;\omega^V)+D(s,a;\omega^D)-\max_{a\in\mathcal{A}}D(s,a;\omega^D)$$

右式第三项恒等于零不能省去，因为这样会导致不唯一性，V 和 D 可以改变参数而不影响输出的 Q。（但是只要 D 最大值保持不变，V 和 D 的值仍然可以随意变化？）

实际实现的时候，用 mean 代替 max 会有更好的效果。对决网络与 DQN 都是对最优动作价值函数 Q_* 的近似，所以对决网络的训练和决策与 DQN完全一样，也可以采用经验回放的方式进行训练，同样也会有 Q 学习的高估问题。

噪声网络（noisy net）可以显著提高 DQN 的表现，将神经网络中的参数 ω 替换为 $\mu+\sigma\circ\xi$，其中 ξ 是随机噪声，它的每个元素独立从标准正态分布 $\mathcal{N}(0,1)$ 中随机抽取，例如某一个全连接层为

$$z=\mathrm{ReLU}(Wx+b)$$

噪声网络把这个全连接层替换为

$$z=\mathrm{ReLU}\left((W^\mu+W^\sigma\circ W^\xi)x+(b^\mu+b^\sigma\circ b^\xi)\right)$$

噪声 DQN 不使用 ϵ-greedy 策略收集经验效果更好，因为噪声也可以带来探索，在做决策时不需要随机性可以把参数 σ 全部设为零。

在实际实现 DQN 的时候应将四种技巧——优先经验回放、双 Q 学习、对决网络、噪声网络——全部用到，使用下列步骤更新参数

1. 用优先经验回放，从数组中抽取一个四元组，记作 (s_j,a_j,r_j,s_{j+1})
2. 用标准正态分布生成 ξ，对噪声 DQN 做正向传播，得到

$$\hat{q}_j=\tilde{Q}(s_j,a_j,\xi;\mu_{\text{now}},\sigma_{\text{now}})$$

3. 用噪声 DQN 选出最优动作

$$\tilde{a}_{j+1} = \arg\max_{a \in \mathcal{A}} \tilde{Q}(s_{j+1}, a, \xi; \mu_{\text{now}}, \sigma_{\text{now}})$$

4. 用标准正态分布生成 ξ'，用目标网络计算价值

$$\tilde{q}^-_{j+1} = \tilde{Q}(s_{j+1}, \tilde{a}_{j+1}, \xi'; \mu^-_{\text{now}}, \sigma^-_{\text{now}})$$

5. 计算 TD 目标和 TD 误差

$$\hat{y}^-_j = r_j + \gamma \cdot \hat{q}^-_{j+1}, \qquad \delta_j = \hat{q}_j - \hat{y}^-_j$$

6. 设 α_μ 和 α_σ 为学习率，做梯度下降更新噪声 DQN 的参数

$$\mu_{\text{new}} \leftarrow \mu_{\text{now}} - \alpha_\mu \cdot \delta_j \cdot \nabla_\mu \tilde{Q}(s_j, a_j, \xi; \mu_{\text{now}}, \sigma_{\text{now}}), \quad \sigma_{\text{new}} \leftarrow \sigma_{\text{now}} - \alpha_\sigma \cdot \delta_j \cdot \nabla_\sigma \tilde{Q}(s_j, a_j, \xi; \mu_{\text{now}},$$

7. 设 $\tau \in (0, 1)$ 是需要手动调整的超参数，做加权平均更新目标网络的参数

$$\mu^-_{\text{new}} \leftarrow \tau \cdot \mu_{\text{new}} + (1-\tau) \cdot \mu^-_{\text{now}}, \quad \sigma^-_{\text{new}} \leftarrow \tau \cdot \sigma_{\text{new}} + (1-\tau) \cdot \sigma^-_{\text{now}}$$

策略学习

策略梯度方法

状态价值函数既依赖于当前状态 s_t，也依赖于策略网络 π 的参数 θ，其定义为

$$V_\pi(s_t) = \mathbb{E}_{A_t \sim \pi(\cdot|s_t;\theta)}[Q_\pi(s_t, A_t)]$$

如果一个策略很好，那么状态价值 $V_\pi(S)$ 的均值应该很大，因此定义目标函数

$$J(\theta) = \mathbb{E}_S[V_\pi(S)]$$

策略梯度定理

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_S\left[\mathbb{E}_{A \sim \pi(\cdot|S;\theta)}\left[\frac{\partial \ln \pi(A \mid S;\theta)}{\partial \theta} \cdot Q_\pi(S, A)\right]\right]$$

上述定理只有在状态 S 服从马尔科夫链的稳态分布 $d(\cdot)$ 的假设下才成立，而且缺少系数 $\frac{1-\gamma^n}{1-\gamma}$，详细推导过程见《DRL》。

随机梯度 $g(s, a; \theta) \triangleq Q_\pi(s, a) \cdot \nabla_\theta \ln \pi(a \mid s; \theta)$ 是策略梯度 $\nabla_\theta J(\theta)$ 的无偏估计，可以做随机梯度上升来更新 θ，但我们计算不出动作价值函数 $Q_\pi(s, a)$，所以需要方法来近似。

利用真实的 u_t 近似 $Q_\pi(s, a)$，可以将随机梯度近似成下列表达式进而更新策略网络参数 θ

$$\tilde{g}(s_t, a_t; \theta) = u_t \cdot \nabla_\theta \ln \pi(a_t \mid s_t; \theta), \qquad \theta_{\text{new}} \leftarrow \theta_{\text{now}} + \beta \cdot \sum_{t=1}^{n} \gamma^{t-1} \cdot \underbrace{u_t \cdot \nabla_\theta \ln \pi(a_t \mid s_t; \theta_{\text{now}})}_{\text{即随机梯度 } \tilde{g}(s_t, a_t; \theta_{\text{now}})}$$

上述简化的推导问题出在状态的稳态概率分布并不知道，需要通过下列严谨推导

$$
\begin{aligned}
\nabla_\theta J(\theta) =& \mathbb{E}_{S_1,A_1}[g(S_1,A_1;\theta)] \\
&+\gamma\cdot\mathbb{E}_{S_1,A_1,S_2,A_2}[g(S_2,A_2;\theta)] \\
&+\gamma^2\cdot\mathbb{E}_{S_1,A_1,S_2,A_2,S_3,A_3}[g(S_3,A_3;\theta)] \\
&+\cdots \\
&+\gamma^{n-1}\cdot\mathbb{E}_{S_1,A_1,S_2,A_2,S_3,A_3,\cdots,S_n,A_n}[g(S_n,A_n;\theta)]
\end{aligned}
$$

通过轨迹 $s_1,a_1,r_1,\quad s_2,a_2,r_2,\quad \cdots,\quad s_n,a_n,r_n$ 对上式中期望做蒙特卡洛近似，得到

$$
\nabla_\theta J(\theta_{\text{now}}) \approx g(s_1,a_1;\theta_{\text{now}})+\gamma\cdot g(s_2,a_2;\theta_{\text{now}})+\cdots+\gamma^{n-1}\cdot g(s_n,a_n;\theta_{\text{now}})
$$

进一步将 $Q_\pi(s_t,a_t)$ 替换为 u_t，得到

$$
g(s_t,a_t;\theta_{\text{now}}) \approx u_t\cdot\nabla_\theta \ln\pi(a_t\mid s_t;\theta_{\text{now}}),\qquad \nabla_\theta J(\theta_{\text{now}})=\sum_{t=1}^{n}\gamma^{t-1}\cdot u_t\cdot\nabla_\theta\ln\pi(a_t\mid s_t;\theta_{\text{now}})
$$

Actor-critic 方法用价值网络 $q(s,a;\omega)$ 来近似随机梯度中未知的动作价值函数 $Q_\pi(s,a)$，策略网络 $\pi(a\mid s;\theta)$ 相当于演员，它基于状态 s 给出动作 a，价值网络 $q(s,a;\omega)$ 相当于评委，它给演员的表现打分，评价在状态 s 的情况下做出动作 a 的好坏程度。不直接将奖励 R 反馈给策略网络的原因是它需要的是利用回报 U 做训练，而不是单纯的奖励。

将策略梯度的无偏估计中的 $Q_\pi(s,a)$ 用价值网络 $q(s,a;\omega)$ 进行近似，得到近似策略梯度

$$
\hat{g}(s,a;\theta)\triangleq \underbrace{q(s,a;\omega)}_{\text{评委的打分}}\cdot\nabla_\theta\ln\pi(a\mid s;\theta)
$$

注意 Actor-critic 方法中评委打分与策略学习的损失函数具有一致性，因此评委打分会越来越高，策略学习的目标是迎合评委的打分标准，所以价值网络的质量很重要。

带基线的策略梯度方法

基于策略梯度公式得出的 REINFORCE 和 actor-critic 方法效果通常不好，只需要微小改动即能大幅提升表现：把 b 作为动作价值函数 $Q_\pi(s,a)$ 的基线（baseline），只要不依赖于动作 A 的任意函数即可：

$$
\nabla_\theta J(\theta)=\mathbb{E}_S\left[\mathbb{E}_{A\sim\pi(\cdot|S;\theta)}\left[(Q_\pi(S,A)-b)\cdot\nabla_\theta\ln\pi(A\mid S;\theta)\right]\right]
$$

不论是让 $b=0$ 还是让 $b=V_\pi(S)$，对期望的结果毫无影响，都会等于 $\nabla_\theta J(\theta)$，因为

$$
\nabla_\theta J(\theta)=\mathbb{E}_S\left[\mathbb{E}_{A\sim\pi(\cdot|S;\theta)}\left[b\cdot\nabla_\theta\ln\pi(A\mid S;\theta)\right]\right]=0
$$

证明如下

$$
\begin{aligned}
\mathbb{E}_{A\sim\pi(\cdot|s;\theta)}\left[b\cdot\frac{\partial \ln \pi(A\mid s;\theta)}{\partial \theta}\right] &= b\cdot\mathbb{E}_{A\sim\pi(\cdot|s;\theta)}\left[\frac{\partial \ln \pi(A\mid s;\theta)}{\partial \theta}\right] \\
&= b\cdot\sum_{a\in\mathcal{A}}\pi(a\mid s;\theta)\cdot\frac{\partial \ln \pi(a\mid s;\theta)}{\partial \theta} \\
&= b\cdot\sum_{a\in\mathcal{A}}\pi(a\mid s;\theta)\cdot\frac{1}{\pi(a\mid s;\theta)}\cdot\frac{\partial \pi(a\mid s;\theta)}{\partial \theta} \\
&= b\cdot\frac{\partial}{\partial \theta}\underbrace{\sum_{a\in\mathcal{A}}\pi(a\mid s;\theta)}_{\text{恒等于1}} \\
&= 0
\end{aligned}
$$

策略梯度可以近似为下面的随机梯度

$$
g_b(s,a;\theta)=[Q_\pi(s,a)-b]\cdot\nabla_\theta \ln \pi(a\mid s;\theta)
$$

得到的随机梯度是策略梯度 $\nabla_\theta J(\theta)$ 的无偏估计，但是 b 的取值会影响方差

$$
\mathrm{Var}=\mathbb{E}_{S,A}\left[\|g_b(S,A;\theta)-\nabla_\theta J(\theta)\|^2\right]
$$

因为 $g_b(S,A)$ 的均值已知为 $\nabla_\theta J(\theta)$，所以想要最小化方差，即最小化

$$
\mathbb{E}_{S,A}[\|g_b(S,A)\|^2]=\mathbb{E}_{S,A}\left[\|(Q_\pi(S,A)-b)\cdot\nabla_\theta \ln \pi(A\mid S;\theta)\|^2\right]=\mathbb{E}_{S,A}\left[(Q_\pi(S,A)-b)^2\cdot\|\nabla_\theta \ln \pi
$$

其中 b 作为与 a 无关的函数，应该接近 $Q_\pi(S,A)$ 关于 A 的均值时方差最小，即
$b=V_\pi(s)=\mathbb{E}_{A\sim\pi(\cdot|s;\theta)}[Q_\pi(s,A)]$。

使用 u_t 近似 $Q_\pi(s,a)$，用神经网络 $v(s;\omega)$ 近似状态价值函数 $V_\pi(s)$，就得到了带基线的 REINFORCE 算法

$$
g(s,a;\theta)=[Q_\pi(s,a)-V_\pi(s)]\cdot\nabla_\theta \ln \pi(a\mid s;\theta),\qquad \tilde{g}(s,a;\theta)=[u-v(s;\omega)]\cdot\nabla_\theta \ln \pi(a\mid s;\theta)
$$

策略网络和价值网络的输入都是状态 s，因此可以让两个神经网络共享卷积网络（信息提取网络）的参数。

策略梯度的无偏估计为

$$
g(s,a;\theta)=[\underbrace{Q_\pi(s,a)-V_\pi(s)}_{\text{优势函数}}]\cdot\nabla_\theta \ln \pi(a\mid s;\theta)
$$

基于上面公式得到的 actor-critic 方法就是 advantage actor-critic。

训练价值网络 $v(s;\omega)$ 的算法是从贝尔曼公式来的

$$
V_\pi(s_t)=\mathbb{E}_{A_t\sim\pi(\cdot|s_t;\theta)}[\mathbb{E}_{S_{t+1}\sim p(\cdot|s_t,A_t)}[R_t+\gamma\cdot V_\pi(S_{t+1})]]
$$

方程左边可以近似成 $v(s_t;\omega)$，右边利用蒙特卡洛近似为 TD 目标 $\hat{y}_t=r_t+\gamma\cdot v(s_{t+1};\omega)$，计算 TD 误差 $\delta_t=\hat{v}_t-\hat{y}_t$，利用梯度下降更新 ω

$$
\omega\leftarrow\omega-\alpha\cdot\delta_t\cdot\nabla_\omega v(s_t;\omega)
$$

训练策略网络的算法也是从贝尔曼公式来的

$$Q_\pi(s_t,a_t)=\mathbb{E}_{S_{t+1}\sim p(\cdot|s_t,a_t)}[R_t+\gamma\cdot V_\pi(S_{t+1})]$$

把近似策略梯度 $g(s_t,a_t;\theta)$ 中的 $Q_\pi(s_t,a_t)$ 替换为上面的期望，得到

$$g(s_t,a_t;\theta)=[Q_\pi(s_t,a_t)-V_\pi(s_t)]\cdot\nabla_\theta\ln\pi(a_t\mid s_t;\theta)=[\mathbb{E}_{S_{t+1}}[R_t+\gamma\cdot V_\pi(S_{t+1})]-V_\pi(s_t)]\cdot\nabla_\theta\ln\pi($$

利用 s_{t+1} 和 r_t 对上面的期望做近似，将状态价值函数 $V_\pi(s)$ 替换为价值网络 $v(s;\omega)$，得到下列策略梯度 $\nabla_\theta J(\theta)$ 的近似用于更新策略网络的参数 θ

$$\tilde{g}(s_t,a_t;\theta)\triangleq[\underbrace{r_t+\gamma\cdot v(s_{t+1};\omega)}_{\text{TD 目标 }\hat{y}_t}-v(s_t;\omega)]\cdot\nabla_\theta\ln\pi(a_t\mid s_t;\theta)$$

上述训练价值网络的算法存在自举，可以使用目标网络 $v(s;\omega^-)$ 计算 TD 目标进行优化，它的结构与价值网络相同，但是参数不同。

策略学习高级技巧

置信域策略优化（trust region policy optimization，TRPO）是一种策略学习方法，相较于策略梯度方法，TRPO 有两个优势：第一，TRPO 表现更稳定，收敛曲线不会剧烈波动，而且对学习率不敏感；第二，TRPO 用更少的经验（即智能体收集到的状态、动作、奖励）就能达到与策略梯度方法相同的表现。

给定变量当前值 θ_{now}，用 $\mathcal{N}(\theta_{\text{now}})$ 表示 θ_{now} 的一个邻域

$$\mathcal{N}(\theta_{\text{now}})=\{\theta\mid\|\theta-\theta_{\text{now}}\|_2\le\Delta\}$$

构造一个函数 $L(\theta\mid\theta_{\text{now}})$ 来替代目标函数 $J(\theta)$ 进行优化，其中 $\mathcal{N}(\theta_{\text{now}})$ 被称作置信域

$$L(\theta\mid\theta_{\text{now}})\text{ 很接近 }J(\theta),\qquad\forall\theta\in\mathcal{N}(\theta_{\text{now}})$$

结合策略学习的目标函数和状态价值函数的等价变换

$$J(\theta)=\mathbb{E}_S[V_\pi(S)],\qquad V_\pi(s)=\sum_{a\in\mathcal{A}}\pi(a\mid s;\theta_{\text{now}})\cdot\frac{\pi(a\mid s;\theta)}{\pi(a\mid s;\theta_{\text{now}})}\cdot Q_\pi(s,a)=\mathbb{E}_{A\sim\pi(\cdot|s;\theta_{\text{now}})}\left[\frac{\pi(A\mid}{\pi(A\mid s}\right.$$

可以得到目标函数 $J(\theta)$ 的等价形式

$$J(\theta)=\mathbb{E}_S\left[\mathbb{E}_{A\sim\pi(\cdot|s;\theta_{\text{now}})}\left[\frac{\pi(A\mid s;\theta)}{\pi(A\mid s;\theta_{\text{now}})}\cdot Q_\pi(s,A)\right]\right]$$

利用真实的轨迹得到目标函数 $J(\theta)$ 等价形式的无偏估计

$$L(\theta\mid\theta_{\text{now}})=\frac{1}{n}\sum_{t=1}^{n}\frac{\pi(a_t\mid s_t;\theta)}{\pi(a_t\mid s_t;\theta_{\text{now}})}\cdot Q_\pi(s_t,a_t)$$

再用两次近似替换 $Q_\pi(s_t,a_t)$

$$Q_\pi(s_t,a_t)\Rightarrow Q_{\pi_{\text{old}}}(s_t,a_t)\Rightarrow u_t$$

得到

$$\tilde{L}(\theta \mid \theta_{\text{now}}) = \frac{1}{n} \sum_{t=1}^{n} \frac{\pi(a_t \mid s_t; \theta)}{\pi(a_t \mid s_t; \theta_{\text{now}})} \cdot u_t$$

求解带约束的最大化问题

$$\max_{\theta} \tilde{L}(\theta \mid \theta_{\text{now}}) \qquad \text{s.t. } \theta \in \mathcal{N}(\theta_{\text{now}})$$

其中的置信域约束可以是球或者 KL 散度

$$\|\theta - \theta_{\text{now}}\|_2 \leq \Delta \quad \text{or} \quad \frac{1}{t} \sum_{i=1}^{t} \text{KL}[\pi(\cdot \mid s_i; \theta_{\text{now}}) \| \pi(\cdot \mid s_i; \theta)] \leq \Delta$$

我们希望策略网络的输出的概率不要集中在一个动作上，因此用熵（Entropy）来衡量概率分布的不确定性

$$\text{Entropy}(p) = -\sum_{i=1}^{t} p_i \cdot \ln p_i$$

策略网络的输出是维度等于 $|\mathcal{A}|$ 的离散概率分布，这个概率分布的熵定义为

$$H(s; \theta) \triangleq \text{Entropy}[\pi(\cdot \mid s; \theta)] = -\sum_{a \in \mathcal{A}} \pi(a \mid s; \theta) \cdot \ln \pi(a \mid s; \theta)$$

因此用熵正则的策略学习可以写作最大化问题

$$\max_{\theta} J(\theta) + \lambda \cdot \mathbb{E}_S[H(s; \theta)]$$

利用策略梯度方法求解该问题，则用下列梯度 $g(\theta)$ 的无偏估计更新网络参数

$$\tilde{g}(s, a; \theta) \triangleq [Q_\pi(s, a) - \lambda \cdot \ln \pi(a \mid s; \theta) - \lambda] \cdot \nabla_\theta \ln \pi(a \mid s; \theta)$$

因为

$$\nabla_\theta [J(\theta) + \lambda \cdot \mathbb{E}_S[H(S; \theta)]] = \mathbb{E}_S[\mathbb{E}_{A \sim \pi(\cdot \mid s; \theta)}[\tilde{g}(S, A; \theta)]]$$

证明如下

$$
\begin{aligned}
\frac{\partial H(s;\theta)}{\partial \theta} &= -\sum_{a\in\mathcal{A}} \frac{\partial [\pi(a\mid s;\theta)\cdot \ln \pi(a\mid s;\theta)]}{\partial \theta} \\
&= -\sum_{a\in\mathcal{A}} \left[\ln \pi(a\mid s;\theta)\cdot \frac{\partial \pi(a\mid s;\theta)}{\partial \theta} + \pi(a\mid s;\theta)\cdot \frac{\ln \pi(a\mid s;\theta)}{\partial \theta} \right] \\
&= -\sum_{a\in\mathcal{A}} \left[\ln \pi(a\mid s;\theta)\cdot \pi(a\mid s;\theta)\cdot \frac{\partial \ln \pi(a\mid s;\theta)}{\partial \theta} + \pi(a\mid s;\theta)\cdot \frac{\ln \pi(a\mid s;\theta)}{\partial \theta} \right] \\
&= -\sum_{a\in\mathcal{A}} \pi(a\mid s;\theta)\cdot [\ln \pi(a\mid s;\theta)+1]\cdot \frac{\partial \ln \pi(a\mid s;\theta)}{\partial \theta} \\
&= -\mathbb{E}_{A\sim\pi(\cdot\mid s;\theta)} \left[[\ln \pi(A\mid s;\theta)+1]\cdot \frac{\partial \ln \pi(A\mid s;\theta)}{\partial \theta} \right]
\end{aligned}
$$

由推导的策略梯度定理有

$$
\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_S \left\{ \mathbb{E}_{A\sim\pi(\cdot\mid s;\theta)} \left[Q_\pi(S,A)\cdot \frac{\partial \ln \pi(A\mid S;\theta)}{\partial \theta} \right] \right\}
$$

因为关于 S 求期望与 θ 无关，所以联合可得

$$
\begin{aligned}
&\frac{\partial}{\partial \theta}[J(\theta)+\lambda\cdot \mathbb{E}_S[H(S;\theta)]] \\
&= \mathbb{E}_S \left\{ \mathbb{E}_{A\sim\pi(\cdot\mid S;\theta)} \left[(Q_\pi(S,A)-\lambda\cdot \ln \pi(A\mid S;\theta)-\lambda)\cdot \frac{\partial \ln \pi(A\mid S;\theta)}{\partial \theta} \right] \right\} \\
&= \mathbb{E}_S \left\{ \mathbb{E}_{A\sim\pi(\cdot\mid S;\theta)} [\tilde{g}(S,A;\theta)] \right\}
\end{aligned}
$$

连续控制

对动作空间做离散化后，就可以应用之前所学的方法训练 DQN 或者策略网络，但是对于高自由度 d 的动作会形成维度灾难。此外，确定策略梯度（deterministic policy gradient，DPG）是最常用的连续控制方法，它是一种 actor-critic 方法，确定策略网络 $\mu(s;\theta)$ 的输出是 d 维的向量 a 作为动作。

本节的确定策略网络属于异策略方法，目标策略即确定策略网络 $\mu(s;\theta_{\text{now}})$，行为策略可以是任意的，如

$$
a = \mu(s;\theta_{\text{old}}) + \epsilon
$$

训练策略网络时应固定价值网络参数 ω，目标函数定为价值网络的打分的期望

$$
J(\theta) = \mathbb{E}_S[q(S,\mu(S;\theta);\omega)]
$$

不管面对什么状态 S，策略网络都应该做出很好的动作，使得平均分 $J(\theta)$ 尽量高，可以用梯度上升来增大 $J(\theta)$，每次用随机变量 S 的一个观测值（记作 s_j）来计算梯度

$$
g_j \triangleq \nabla_\theta q(s_j,\mu(s_j;\theta);\omega)
$$

利用链式法则求出确定策略梯度 g_j

$$
\nabla_\theta q(s_j,\mu(s_j;\theta);\omega) = \nabla_\theta \mu(s_j;\theta)\cdot \nabla_a q(s_j,\hat{a}_j;\omega), \quad \text{其中 } \hat{a}_j = \mu(s_j;\theta)
$$

训练价值网络时需要使价值网络 $q(s,a;\omega)$ 的预测越来越接近真实价值函数 $Q_\pi(s,a)$，因此需要用到 TD 算法，固定参数 θ，首先让价值网络做预测

$$\hat{q}_j = q(s_j, a_j; \omega) \quad 和 \quad \hat{q}_{j+1} = q(s_{j+1}, \mu(s_{j+1};\theta); \omega)$$

然后计算 TD 目标 $\hat{y}_j = r_j + \gamma \cdot \hat{q}_{j+1}$，以缩小 TD 误差为目标进行优化。

实际运行中 DPG 效果并不好，因为也存在类似 DQN 高估的两个问题，可以使用目标网络计算 TD 目标缓解高估

$$\hat{y}_j = r_j + \gamma \cdot q(s_{j+1}, \hat{a}_{j+1}; \omega^-), \qquad 其中\ \hat{a}_{j+1} = \mu(s_{j+1};\theta^-)$$

但是实验表明高估仍然很严重，更好的解决方案是截断双 Q 学习（clipped double Q-learning），这种方法使用两个价值网络和一个策略网络以及各自对应的目标网络，共六个神经网络。用目标网络计算动作

$$\hat{a}^-_{j+1} = \mu(s_{j+1};\theta^-)$$

然后用两个目标价值网络计算，取两者较小者为 TD 目标

$$\hat{y}_{j,1} = r_j + \gamma \cdot q(s_{j+1}, \hat{a}^-_{j+1}; \omega^-_1), \quad \hat{y}_{j,2} = r_j + \gamma \cdot q(s_{j+1}, \hat{a}^-_{j+1}; \omega^-_2)$$

可以进一步改进，通过往动作中加入噪声使结果更加稳健

$$\hat{a}^-_{j+1} = \mu(s_{j+1};\theta^-) + \xi$$

其中 ξ 是个随机向量，每一个元素独立从截断正态分布（clipped normal distribution）中抽取，该分布为 $\mathcal{CN}(0, \sigma^2, -c, c)$，限制在 $-c$ 和 c 之间防止噪声过大。

此外，实验表明，应当让策略网络 μ 以及三个目标网络的更新慢于价值网络 q，确保评价的正确性。

使用上述三种技巧的算法被称作双延时确定策略梯度（twin delayed deep deterministic policy gradient，TD3）。

随机高斯策略网络利用多元正态分布的概率密度函数作为策略网络

$$\pi(a \mid s;\theta) = \prod_{i=1}^{d} \frac{1}{\sqrt{6.28 \cdot \exp[\rho_i(s;\theta)]}} \cdot \exp\left(-\frac{[a_i - \mu_i(s;\theta)]^2}{2 \cdot \exp[\rho_i(s;\theta)]}\right)$$

构造辅助网络

$$f(s,a;\theta) = -\frac{1}{2}\sum_{i=1}^{d}\left(\rho_i(s;\theta) + \frac{[a_i - \mu_i(s;\theta)]^2}{\exp[\rho_i(s;\theta)]}\right)$$

辅助网络与策略网络的关系为

$$f(s,a;\theta) = \ln \pi(a \mid s;\theta) + \text{Constant}$$

根据策略梯度的蒙特卡洛近似可以计算

$$g = Q_\pi(s,a) \cdot \nabla_\theta \ln \pi(a \mid s;\theta) = Q_\pi(s,a) \cdot \nabla_\theta f(s,a;\theta)$$

其中的 $Q_\pi(s,a)$ 未知，可以采用 REINFORCE 用实际观测的折扣回报代替 $Q_\pi(s,a)$，或者 actor-critic 用价值网络近似 Q_π，采用带基线的方法效果更好。

对状态的不完全观测

在实际应用中，完全观测假设存在缺陷，对于不完全观测的强化学习问题，应当记忆过去的观测，用所有已知的信息做决策，这也是人类解决不完全观测问题的方式。所以可以用 $o_{1:t}$ 来代替状态 s，作为策略网络的输入

$$\pi(a_t; o_{1:t}; \theta)$$

循环神经网络（recurrent neural network，RNN）可以将一个序列映射到一个特征向量（其实是一个序列，但最后的特征向量包含了此前事件的信息），可以利用此特性搭建策略网络、DQN 和价值网络。

模仿学习

模仿学习（imitation learning）不是强化学习，虽然两者的目的都是学习策略网络从而控制智能体，但是模仿学习的原理是向人类专家学习，目标是让策略网络做出的决策与人类专家相同；而强化学习的原理是利用环境反馈的奖励改进策略，目标是让累计奖励（即回报）最大化。

行为克隆（behavior cloning）是最简单的模仿学习，利用（状态，动作）二元组构成的数据集 $\mathcal{X} = \{(s_1,a_1), \cdots, (s_n,a_n)\}$ 作监督学习，连续控制问题和离散控制问题的损失函数分别为

$$L(s,a;\theta) \triangleq \frac{1}{2}[\mu(s;\theta) - a]^2, \qquad H(\bar{a}, f) \triangleq -\sum_{i=1}^{\mathcal{A}} \bar{a}_i \cdot \ln f_i$$

行为克隆训练出的策略网络通常效果不佳。人类不会探索奇怪的状态和动作，因此 数据集上的状态和动作缺乏多样性。在数据集上做完行为克隆之后，智能体面对真实的 环境,可能会见到陌生的状态，此时做出的决策可能会很糟糕。行为克隆存在“错误累加”的缺陷，加入当前的动作 a_t 不够好，那么下一时刻的状态 s_{t+1} 可能会比较罕见，于是下一个动作 a_{t+1} 会很差，这又导致状态 s_{t+1} 非常奇怪，使得动作 a_{t+1} 更糟糕。虽然行为克隆效果不如强化学习，但是行为克隆的成本低。可以先用行为克隆初始化策略网络（而不是随机初始化），然后再做强化学习，这样可以减小对物理世界的有害影响。

逆向强化学习（inverse reinforcement learning，IRL）曾经非常有名，但现在已不常用了，下节介绍的 GAIL 更简单效果更好。

IRL 的基本设定：第一，IRL 假设智能体可以与环境交互，环境会根据智能体的动作更新状态，但是不会给出奖励；第二，IRL 假设人类专家的策略 $\pi^*(a \mid s)$ 是一个黑箱，人类学习策略的方式与强化学习相同，都是最大化回报（即累计奖励）的期望

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{S_t, A_t, \cdots, S_n, A_n} \left[\sum_{k=t}^{n} \gamma^{k-t} \cdot R^*(S_k, A_k) \right]$$

IRL 的目的是学到一个策略网络 $\pi(a \mid s; \theta)$，模仿人类专家的黑箱策略 $\pi^*(a \mid s)$。首先从 $\pi^*(a \mid s)$ 中学习其隐含的奖励函数 R^*，然后利用奖励函数做强化学习，得到策略网络的参数 θ。

生成判别模仿学习（generative adversarial imitation learning，GAIL）需要让智能体与环境交互，但是无法从环境获得奖励，其目标是学习一个策略网络，使得判别器无法区分一条轨迹是策略网络的决策还是人类专家的决策。

GAIL 的设计基于生成判别网络（generative adversarial network，GAN），生成器记作 $x = G(s; \theta)$，s 为简单分布抽取的向量，判别器记作 $\hat{p} = D(x; \phi)$，输出为 0 到 1之间的概率值。生成器的损失函数可以用交叉熵

$$E(s; \theta) = \ln \left[1 - \underbrace{D(x; \phi)}_{\text{越大越好}} \right]; \qquad \text{s.t. } x = G(s; \theta)$$

判别器的损失函数也用交叉熵

$$F(x^{\text{real}}, x^{\text{fake}}; \phi) = \ln \left[1 - \underbrace{D(x^{\text{real}}; \phi)}_{\text{越大越好}} \right] + \ln \underbrace{D(x^{\text{fake}}; \phi)}_{\text{越小越好}}$$

在 GAIL 的生成器训练中，用策略网络 $\pi(a \mid s; \theta_{\text{now}})$ 控制智能体与环境交互，得到其轨迹，将 $u_t = \ln D(s_t, a_t; \phi)$ 作为第 t 步的回报，于是可以用 TRPO 来更新策略网络，定义目标函数

$$\tilde{L}(\theta \mid \theta_{\text{now}}) \triangleq \frac{1}{n} \sum_{t=1}^{n} \frac{\pi(a_t \mid s_t; \theta)}{\pi(a_t \mid s_t; \theta_{\text{now}})} \cdot u_t$$

求解下面的带约束的最大化问题，得到新的参数

$$\theta_{\text{new}} = \arg\max_{\theta} \tilde{L}(\theta \mid \theta_{\text{now}}); \qquad \text{s.t. } \text{dist}(\theta_{\text{now}}, \theta) \le \Delta$$

训练判别器时从训练数据中均匀抽样一条轨迹 τ^{real}，用策略网络控制智能体与环境交互得到一条轨迹 τ^{fake}，定义损失函数并作梯度下降更新参数 ϕ

$$F(\tau^{\text{real}}, \tau^{\text{fake}}; \phi) = \underbrace{\frac{1}{m} \sum_{t=1}^{m} \ln \left[1 - D(s_t^{\text{real}}, a_t^{\text{real}}; \phi) \right]}_{D \text{ 的输出越大，该项越小}} + \underbrace{\frac{1}{n} \sum_{t=1}^{n} \ln D(s_t^{\text{fake}}, a_t^{\text{fake}}; \phi)}_{D \text{ 的输出越小，该项越小}}$$

多智能体强化学习

并行计算

并行计算需要在计算机集群上完成。一个集群有很多处理器和内存条，它们被划分到多个节点（compute node）上。MapReduce 属于 client-server 架构，有一个节点作为中央服务器，其余节点作为 worker，受服务器控制。

服务器可以与 worker 节点做通信传输数据（但是 worker 节点之间不能相互通信）。一 种通信方式是广播（broadcast），即服务器将同一条信息同时发送给所有 worker 节点。映射（map）操作让所有 worker 节点同时并行做计算，最常用的通信操作是规约（reduce）。这种操作可以把 worker 节点上的数据做归并，并且传输到服务器上。

数据并发（data parallelism）是将数据集 $(x_1, y_1), \cdots, (x_n, y_n)$ 划分到 m 个 worker 节点上，每个节点上存一部分数据；模型并发（model parallelism）即将模型参数 ω 划分到 m 个worker 节点上，每个节点有全部数据。

钟表时间（wall-clock time），也叫 elapsed real time，意思是程序实际运行的时间；处理器时间（CPU time 或 GPU time）是所有处理器运行时间的总和。利用加速比（speedup ratio）衡量并行计算带来的速度提升

$$\text{加速比} = \frac{\text{使用一个节点所需的钟表时间}}{\text{使用 } m \text{ 个节点所需的钟表时间}}$$

通信量（communication complexity）的意思是有多少个比特或者浮点数在服务器与 worker 节点之间传输。延迟（latency）是由计算机网络的硬件和软件系统决定的。通信时间主要由通信量和延迟造成

$$\text{通信时间} \approx \frac{\text{通信量}}{\text{带宽}} + \text{延迟}$$

同步算法（synchronous algorithm）要求每一轮都必须等待所有节点完成计算，这势必导致“短板效应”，即任务所需时间取决于最慢的节点。Straggler effect 意思是一个节点的速度远慢于其余节点，导致整个系统长时间处于空闲状态，等待最慢的节点。

异步算法（asynchronous algorithm）中一个 worker 节点无需等待其余节点完成计算或通信。当一个 worker 节点完成计算，它立刻跟 server 通信，然后开始下一轮的计算。异步梯度下降算法与标准的梯度下降是不等价的，在理论上，异步梯度下降的收敛速度慢于同步算法，即需要更多的计算量才能达到相同的精度。但是实践中异步梯度下降远比同步算法快（用钟表时间衡量），这是因为异 步算法无需等待，worker 节点几乎不会空闲，利用率很高。

异步并行双 Q 学习中服务器和 worker 都存储 DQN 的参数，worker 节点有自己的目标网络，而服务器上不存储目标网络，每个 worker 节点有自己的环境，用 DQN 控制智能体与环境交互，收集经验，把 (s, a, r, s') 这样的四元组存储到本地的经验回放数组。

第 k 号 worker 节点重复下面的步骤：

1. 向服务器发出请求，索要最新的 DQN 参数，把接收到的参数记作 ω_{new}
2. 更新本地的目标网络

$$\omega^{-}_{\text{new}} \leftarrow \tau \cdot \omega_{\text{new}} + (1-\tau) \cdot \omega^{-}_{\text{now}}$$

3. 在本地做经验回放，计算本地梯度
   (a). 从本地的经验回放数组中随机抽取 b 个四元组 $(s_1, a_1, r_1, s_1'), \cdots, (s_b, a_b, r_b, s_b')$
   (b). 用双 Q 学习计算 TD 目标

$$\hat{y}_j = r_j + \gamma \cdot Q(s_j', a_j'; \omega_{\text{new}}^-), \quad \text{其中 } a_j' = \arg\max_a Q(s_j', a; \omega_{\text{new}})$$

   (c). 定义目标函数

$$L(\omega) =^{\triangleq} \frac{1}{2b} \sum_{j=1}^{b} [Q(s_j, a_j; \omega) - \hat{y}_j]^2$$

   (d). 计算梯度

$$\tilde{g}^k = \nabla_\omega L(\omega_{\text{new}})$$

4. 把计算出的梯度 $\tilde{g}^k$ 发送给服务器

多智能体系统

多智能体系统（multi-agent system，MAS）相较于单智能体系统（single-agent system，SAS），有 m 个智能体，智能体共享环境，智能体之间会相互影响。并行强化学习的设定是 m 个单智能体系统的并集，可以视作 MAS 的一种特例。

多智能体强化学习（multi-agent reinforcement learning，MARL）是指让多个智 能体处于相同的环境中，每个智能体独立与环境交互，利用环境反馈的奖励改进自己的 策略，以获得更高的回报（即累计奖励）。

多智能体系统有四种常见设定

1. 完全合作关系（fully cooperative）：智能体的利益一致，获得的奖励相同。
2. 完全竞争关系（fully competitive）：一方的收益是另一方的损失。
3. 合作竞争的混合（mixed cooperative & competitive）：智能体分成多个群组，组内智能体是合作关系，它们的奖励相同，组间是竞争关系，两组的奖励是负相关的。
4. 利己主义（self-interested）：智能体只想最大化自身的累计奖励，而不在乎他人收益或者受损。

多智能体系统的状态价值函数可以写成

$$V_\pi^i(s) = \mathbb{E}_A[Q_\pi^i(s, A)] = \sum_{a^1 \in \mathcal{A}^1} \sum_{a^2 \in \mathcal{A}^2} \cdots \sum_{a^m \in \mathcal{A}^m} \pi(a \mid s; \theta^1, \cdots, \theta^m) \cdot Q_\pi^i(s, a)$$

合作关系设定下的多智能体强化学习

多智能体强化学习（MARL）大多假设所有的局部观测的总和构成全局状态

$$S = [O^1, O^2, \cdots, O^m]$$

本章所采用的符号如下图所示

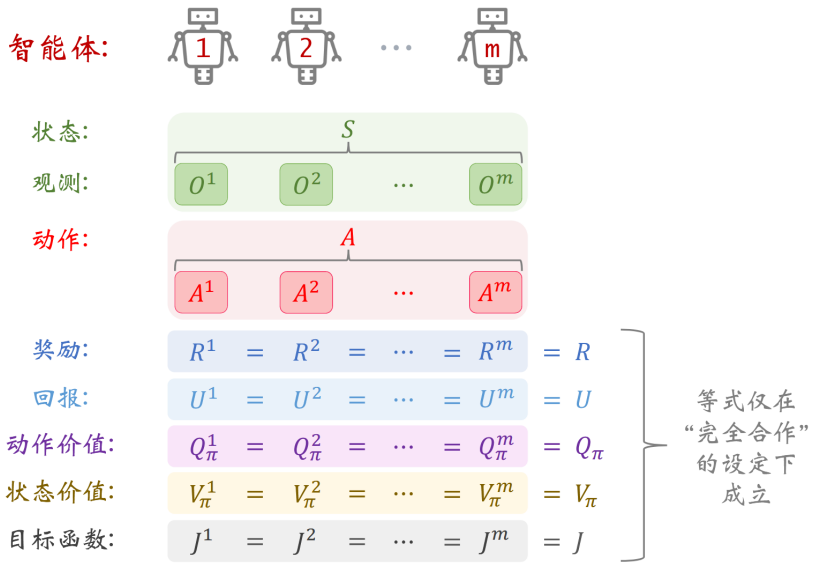

图 15.1: 多智能体强化学习 (MARL) 在“完全合作关系”设定下的符号。

完全合作关系的意思是所有智能体的利益是一致的，每完成一个任务，团队成员（即智能体）获得相同的奖励（总体奖励相同，该奖励可能是直接从环境中观测到的，也可能是所有智能体本地的奖励 $\tilde{r}_t^i$ 的加和），所以大家的 R, U, Q_π, V_π 全都是一样的（此处 Q_π 应为团队总价值），通常来说，团队成员有分工合作，因此每个成员的策略是不同的，即 $\theta^i \neq \theta^j$。

做策略学习时（即学习策略网络参数 $\theta^1, \cdots, \theta^m$），所有智能体都有一个共同目标函数以及对应的优化问题

$$J(\theta^1, \cdots, \theta^m) = \mathbb{E}_S[V_\pi(S)], \qquad \max_{\theta^1, \cdots, \theta^m} J(\theta^1, \cdots, \theta^m)$$

多智能体完全合作关系设定下的演员-评委方法（multi-agent cooperative advantage actor-critic，MAC-A2C）中所有智能体共用一个价值网络 $v(s;\omega)$，共用一个策略网络的结构。价值网络的训练与单智能体 A2C 的 TD 算法完全相同。

训练策略网络时需要用到合作关系 MARL 的策略梯度定理：设基线 b 为不依赖于 $A = [A^1, \cdots, A^m]$ 的函数，则有

$$\nabla_{\theta^i} J(\theta^1, \cdots, \theta^m) = \mathbb{E}_{S,A}\left[(Q_\pi(S, A) - b) \cdot \nabla_{\theta^i} \ln \pi(A^i \mid S; \theta^i)\right]$$

期望中动作 A 的概率质量函数为

$$\pi(A \mid S; \theta^1, \cdots, \theta^m) \triangleq \pi(A^1 \mid S; \theta^i) \times \cdots \times \pi(A^m \mid S; \theta^m)$$

将基线设置为状态价值 $b = V_\pi(s)$，定义

$$g^i(s, a; \theta^i) \triangleq (Q_\pi(s, a) - V_\pi(s)) \cdot \nabla_{\theta^i} \ln \pi(a^i \mid s; \theta^i)$$

易知 $g^i(s, a; \theta^i)$ 是策略梯度的无偏估计，再将 $Q_\pi(s_t, a_t)$ 近似为 $r_t + \gamma \cdot v(s_{t+1}; \omega)$，把 $V_\pi(s_t; \omega)$ 近似为 $v(s_t; \omega)$，则近似策略梯度 $g^i(s_t, a_t; \theta^i)$ 可以进一步近似成

$$\tilde{g}^i(s_t, a_t; \theta^i) \triangleq \underbrace{\left(r_t + \gamma \cdot v(s_{t+1}; \omega) - v(s_t; \omega)\right)}_{\text{对 } Q_\pi(s_t, a_t) - V_\pi(s_t) \text{ 的近似}} \cdot \nabla_{\theta^i} \ln \pi(a_t^i \mid s_t; \theta^i)$$

在常见的 MARL 设定下，第 i 号智能体只知道 o^i，而观测不到全局状态，会导致缺乏全局状态 s 做决策、训练价值网络、训练策略网络。解决办法有两个：

1. 智能体共享观测，也即中心化（centralized），中心化的通讯会让训练和决策的速度变慢。
2. 对策略网络和价值函数做近似，也即去中心化（decentralized），通常使用 $\pi(a^i \mid o^i;\theta^i)$ 替代 $\pi(a^i \mid s;\theta^i)$，甚至可以进一步用 $v(o^i;\omega^i)$ 代替 $v(s;\omega)$，缺点是不完全信息造成训练不收敛、做出错误决策。

中心化训练+中心化决策：
价值网络与策略网络均在中央控制器（central controller），智能体只收集观测和执行动作，其训练和决策完全按照 MAC-A2C 的算法实现。

去中心化训练+去中心化决策：
每个智能体上部署一个策略网络和一个价值网络，智能体之间不共享参数，假设所有智能体的奖励都是相通的，而且每个智能体都能观测到奖励 r，这样 MAC-A2C 就变成了标注的 A2C，其本质就是单智能体强化学习（SARL）。

中心化训练+去中心化决策：
价值网络和目标网络部署到中央控制器上，策略网络部署在各智能体上，只需要训练时利用中央控制器计算 TD 误差并广播到各智能体上更新策略网络参数，决策时不再需要中央控制器。

非合作关系设定下的多智能体强化学习

第 i 号智能体的目标函数是状态价值的期望

$$J^i(\theta^1,\cdots,\theta^m)=\mathbb{E}_S\big[V_\pi^i(S)\big]$$

各个智能体的目标函数是不相同的，并且都依赖于所有智能体的策略网络参数 $\theta^1,\cdots,\theta^m$。

非合作关系设定下，策略学习的收敛标准是纳什均衡，即每个智能体都以最优的方式来应对其他各方的策略，谁也没有动机去单独改变自己的策略。在实验中，如果所有智能体的平均回报都不再变化，就可以认为达到了纳什均衡。

有两种策略学习的方法 $\mathcal{M}_+$ 和 $\mathcal{M}_-$，利用它们训练出的策略网络参数分别记作 $\theta_+^1,\cdots,\theta_+^m$ 和 $\theta_-^1,\cdots,\theta_-^m$。在合作关系的设定下，把两种策略的平均回报记作 J_+ 和 J_-，如果 $J_+>J_-$，则说明 $\mathcal{M}_+$ 比 $\mathcal{M}_-$ 好；在非合作关系的设定下，以捕食者-猎物游戏为例，将两种方法训练出的策略网络记作 $\pi(a\mid s,\theta_+^{\text{predator}}),\pi(a\mid s;\theta_+^{\text{prey}})$　$\pi(a\mid s;\theta_-^{\text{predator}}),\pi(a\mid s;\theta_-^{\text{prey}})$，然后用一种方法训练出的捕食者与另一种方法训练出的猎物对决：

$$\pi(a\mid s;\theta_+^{\text{predator}})\text{ 对决 }\pi(a\mid s;\theta_-^{\text{prey}})\qquad\pi(a\mid s;\theta_-^{\text{predator}})\text{ 对决 }\pi(a\mid s;\theta_+^{\text{prey}})$$

如果我们的目标是学出强大的捕食者，则记录下两方捕食者的平均回报，记作 J_+^{predator}、J_-^{predator}，两者的大小可以反映出 $\mathcal{M}_+$ 和 $\mathcal{M}_-$ 的优劣。

在非合作关系设定下的多智能体 A2C 方法（multi-agent non-cooperative advantage actor-critic，MAN-A2C）假设第 i 号智能体能获取所有智能体的观测 $s=[o^1,\cdots,o^m]$（违背直觉，在训练阶段，有一个中心化的训练控制器或者说“上帝”，它可以同时观察到所有智能体的状态。），每个智能体各自对应一个价值网络 $v(s;\omega^i)$。利用下列非合作关系 MARL 的策略梯度定理计算梯度后更新策略网络，采用 $b=V_\pi^i(s)$ 作为定理中的基线

$$\nabla_{\theta^i}J^i(\theta^1,\cdots,\theta^m)=\mathbb{E}_{S,A}\Big[\big(Q_\pi^i(S,A)-b\big)\cdot\nabla_{\theta^i}\ln\pi(A^i\mid S;\theta^i)\Big]$$

中心化训练+中心化决策：
中央控制器部署了所有 m 个价值网络和策略网络，智能体只负责与环境进行交互。

去中心化训练+去中心化决策：
每个智能体上部署一个策略网络和一个价值网络，其本质是单智能体强化学习。

中心化训练+去中心化决策：
与完全中心化的 MAN-A2C 相比唯一的区别是策略网络部署在各个智能体上，并且对策略网络做近似 $\pi(a^i\mid s;\theta^i)\to\pi(a^i\mid o^i;\theta^i)\quad\forall i=1,\cdots,m$。

多智能体深度确定策略梯度（multi-agent deep deterministic policy gradient，MADDPG）用于连续控制，其架构是中心化训练+去中心化决策。系统里每个智能体对应一个策略网络 $\mu(o^i;\theta^i)$ 和 一个价值网络 $q(s,a;\omega^i)$，注意第 i 号价值网络的输入是所有动作和状态 s，因此可以看作一种 actor-critic 方法。

MADDPG 的训练算法与单智能体 DPG 非常类似，用确定策略梯度更新策略网络，用 TD 算法更新价值网络。可以用三种方法改进 MADDPG：

1. 用 TD3 的三种技巧改进训练的算法：用截断双 Q 学习（clipped double Q-learning）训练价值网络 $q(s,a;\omega^i),\quad\forall i=1,\cdots,m$；往训练算法第一步中的 $\hat{a}_{t+1}^{i-}$ 加入噪声；减小更新策略网络和目标网络的频率。
2. 在策略网络和价值网络中使用 RNN，记忆历史观测。
3. 在价值网络的结构中使用注意力机制。

注意力机制与多智能体强化学习

自注意力机制能将长度为 m 的序列 $(x^1,\cdots,x^m)$ 处理后输出长度为 m 的序列 $(c^1,\cdots,c^m)$，并且长度 m 是可以变化而参数量是不变的，此外输出 c^i 不仅依赖于 x^i，还依赖于所有的输入向量 $(x^1,\cdots,x^m)$。

不使用自注意力的状态价值网络输入是所有智能体的观测的连接，输出是实数 $\hat{v}^i=v\big([o^1,\cdots,o^m];\omega^i\big)$，这样简单的神经网络结构存在不足：智能体数量 m 增大时神经网络的参数量增大过快；当 m 很大时，并非所有智能体的观测 $o^1,\cdots,o^m$ 都与第 i 号智能体密切相关。

自注意力机制在非合作关系的 MARL 中普遍适用。如果系统架构使用中心化训练，那么 m 个价值网络可以用一个神经网络实现，其中使用自注意力层。如果系统 架构使用中心化决

策，那么 m 个策略网络也可以实现成一个神经网络，其中使用自注意力层。在 m 较大的情况下，使用自注意力层对效果有较大的提升。

应用与展望

AlphaGo 与蒙特卡洛树搜索

价值学习（value-based）和策略学习（policy-based）都是无模型的强化学习（model-free），蒙特卡洛树搜索（Monte Carlo Tree Search，MCTS）是一种基于模型的强化学习方法（model-based）。

无模型方法不去尝试理解环境的内在工作原理（即状态是如何转移的，奖励是如何产生的）。它的目标是直接学习一个“行为准则”，这个准则告诉智能体在什么状态下应该做什么动作。
基于模型的方法试图先构建一个对环境的内部模拟，即模型（Model）。这个模型主要包含两个部分：状态转移模型 (Transition Model)：预测在状态 s 执行动作 a 后，会转移到哪个新状态 s'。即 $P(s' \mid s, a)$；奖励模型 (Reward Model)：预测在状态 s 执行动作 a 并转移到 s' 后，会获得多少奖励。即 $R(s, a, s')$。

假设已经训练好了策略网络 $\pi(a \mid s; \theta)$ 和价值网络 $v(s; \omega)$，AlphaGo 真正跟人下棋的时候，做决策的不是策略网络或者价值网络，而是 MCTS，MCTS 不需要训练，可以直接做决策。其基本思想是向前看，假设当前有多种较好的动作，每次模拟时从中选一种然后将游戏进行到底，从而知晓胜负，然后统计每种动作的胜负频率，选择胜率最大的动作。

MCTS 每次模拟分为四个步骤：

1. 选择（selection）：通过两个指标判断动作 a 的质量，分别是动作 a 的胜率和策略网络给动作 a 的概率值

   $$\text{score}(a) \triangleq Q(a) + \frac{\eta}{1 + N(a)} \cdot \pi(a \mid s; \theta)$$

   其中 η 是超参数，$N(a)$ 是动作 a 被访问过的次数，$Q(a)$ 是之前 $N(a)$ 次模拟算出来的动作价值，主要由胜率和价值函数决定，其初始值为 0。
2. 扩展（expansion）：AlphaGo 用策略网络模拟对手，根据策略网络随机抽样一个动作

   $$a'_t \sim \pi(\cdot \mid s'_t; \theta)$$

   其中 s'_t 是对手观测到的棋盘上的格局，a'_t 是（假想）对手选择的动作，因为对手的动作就是 AlphaGo 的新状态（通过围棋的对称性，获取尽量正确的状态转移函数 $p(s_{k+1} \mid s_k, a_k)$，形成模拟器）。
3. 求值（evaluation）：从状态 s_{t+1} 开始，双方都用策略网络 π 做决策，直到分出胜负，如果 AlphaGo 胜利，则 $r = +1$，否则 $r = -1$。我们想要获得对状态 s_{t+1} 的评价，奖励 r 是模拟获得的胜利，是对 s_{t+1} 很可靠的评价，但是随机性太大；价值网络的评估 $v(s_{t+1}; \omega)$ 没有 r 可靠，但是更稳定，随机性小。最终记 $V(s_{t+1}) \triangleq \frac{r + v(s_{t+1}; \omega)}{2}$ 为状态 s_{t+1} 的评价。

4. 回溯（backup）：在 a_t 后所有 S_{t+1} 状态的评价取均值作为 $Q(a_t)$。

在经过多次模拟后，MCTS 通过 $a_t = \arg\max_a N(a)$ 做出真正的决策。

AlphaGo 2016版的训练分为三步：第一，用行为克隆从人类棋谱中学习策略网络；第二，让两个策略网络自我博弈，用REINFORCE 算法改进策略网络；第三，基于已经训练好的策略网络，训练价值网络

AlphaGo Zero 训练策略网络时不再从人类棋谱学习，也不用 REINFORCE 算法，而是向 MCTS 学习。

现实世界中的应用

1. 神经网络结构搜索（neural architecture search，NAS）的意思是自动寻找最优的神经网络结构，代替手动设计的神经网络。
2. 自动生成 SQL 语句。
3. 推荐系统：据此书作者了解，强化学习尚未在工业界的推荐系中统取得显著效果，学术论文中报告的结果与工业界线上实测结果不一致。虚拟淘宝系统。
4. 网约车调度：值得注意的是,在学习的过程中要用正则项，使得价值网络是光滑的。当状态 s = (地点, 时间) 中的地点、时间发生较小的变化时，价值网络的输出不应该剧烈变化。

监督学习假设用户兴趣点（环境）是固定的，推荐系统只会拟合用户的喜好，推荐相似的物品。而强化学习则假设用户的兴趣点可以被改变，学出的推荐策略会发掘用户新的兴趣点。

强化学习制约：

1. 所需的样本数量过大。
2. 探索阶段代价太大。
3. 超参数的影响非常大：拿激活函数来说，在监督学习中，在隐层中用不同的激活函数（比如 ReLU、Leaky ReLU）对结果影响很小，因此总是用 ReLU 就可以。但是在深度强化学习中，隐层激活函数对结果的影响很大；有时 ReLU 远好于 Leaky ReLU；而有时 Leaky ReLU 远好于 ReLU。
4. 稳定性极差：仅仅更改随机数种子，就会导致训练的效果有天壤之别，甚至会出现完全不收敛的情形，哪怕代码和超参数都是对的。

贝尔曼方程

根据回报的定义 $U_t = \sum_{k=t}^{n} \gamma^{k-t} \cdot R_k$，易知 $U_t = R_t + \gamma \cdot U_{t+1}$。用符号 $\mathcal{S}_{t+1:} = \{S_{t+1}, S_{t+2}, \cdots\}$ 和 $\mathcal{A}_{t+1:} = \{A_{t+1}, A_{t+2}, \cdots\}$ 表示从 $t+1$ 时刻起所有的状态和动作随机变量，再根据动作价值函数 Q_π 的定义

$$Q_\pi(s_t, a_t) = \mathbb{E}_{\mathcal{S}_{t+1:}, \mathcal{A}_{t+1:}}[U_t \mid S_t = s_t, A_t = a_t]$$

那么有

$$
\begin{aligned}
Q_\pi(s_t, a_t) &= \mathbb{E}_{S_{t+1:}, A_{t+1:}}[R_t + \gamma \cdot U_{t+1} \mid S_t = s_t, A_t = a_t] \\
&= \mathbb{E}_{S_{t+1:}, A_{t+1:}}[R_t \mid S_t = s_t, A_t = a_t] + \gamma \cdot \mathbb{E}_{S_{t+1:}, A_{t+1:}}[U_{t+1} \mid S_t = s_t, A_t = a_t]
\end{aligned} \tag{A.1}
$$

假设 R_t 是 S_t, A_t, S_{t+1} 的函数，则

$$
\mathbb{E}_{S_{t+1:}, A_{t+1:}}[R_t \mid S_t = s_t, A_t = a_t] = \mathbb{E}_{S_{t+1}}[R_t \mid S_t = s_t, A_t = a_t] \tag{A.2}
$$

而等式 (A.1) 右侧 U_{t+1} 的期望可以写作

$$
\begin{aligned}
&\mathbb{E}_{S_{t+1:}, A_{t+1:}}[U_{t+1} \mid S_t = s_t, A_t = a_t] \\
&= \mathbb{E}_{S_{t+1}, A_{t+1}}\left[\mathbb{E}_{S_{t+2:}, A_{t+2:}}[U_{t+1} | S_{t+1}, A_{t+1}] \middle| S_t = s_t, A_t = a_t\right] \\
&= \mathbb{E}_{S_{t+1}, A_{t+1}}[Q_\pi(S_{t+1}, A_{t+1}) | S_t = s_t, A_t = a_t]
\end{aligned} \tag{A.3}
$$

由公式 (A.1)、(A.2)、(A.3) 可得

定理 A.1 贝尔曼方程（将 Q_π 表示成 Q_π）：假设 R_t 是 S_t, A_t, S_{t+1} 的函数，那么

$$
Q_\pi(s_t, a_t) = \mathbb{E}_{S_{t+1}, A_{t+1}}[R_t + \gamma \cdot Q_\pi(S_{t+1}, A_{t+1}) \big| S_t = s_t, A_t = a_t]
$$

由于 $V_\pi(S_{t+1}) = \mathbb{E}_{A_{t+1}}[Q(S_{t+1}, A_{t+1})]$，由定理 A.1 可得定理 A.2

定理 A.2 贝尔曼方程（将 Q_π 表示成 V_π）：假设 R_t 是 S_t, A_t, S_{t+1} 的函数，那么

$$
Q_\pi(s_t, a_t) = \mathbb{E}_{S_{t+1}}[R_t + \gamma \cdot V_\pi(S_{t+1}) \big| S_t = s_t, A_t = a_t]
$$

由于 $V_\pi(S_t) = \mathbb{E}_{A_t}[Q(S_t, A_t)]$，由定理 A.2 可得定理 A.3

定理 A.3 贝尔曼方程（将 V_π 表示成 V_π）：假设 R_t 是 S_t, A_t, S_{t+1} 的函数，那么

$$
V_\pi(s_t) = \mathbb{E}_{A_t, S_{t+1}}[R_t + \gamma \cdot V_\pi(S_{t+1}) \big| S_t = s_t]
$$

设最优策略函数为 $\pi^* = \arg\max_\pi Q_\pi(s, a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$。由贝尔曼方程可得

$$
Q_{\pi^*}(s_t, a_t) = \mathbb{E}_{S_{t+1}, A_{t+1}}[R_t + \gamma \cdot Q_{\pi^*}(S_{t+1}, A_{t+1}) \big| S_t = s_t, A_t = a_t]
$$

由最优动作价值函数的定义 $Q_*(s, a) \triangleq \max_\pi Q_\pi(s, a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$，所以 $Q_{\pi^*}(s_t, a_t)$ 就是 $Q_*(s, a)$，于是

$$
Q_*(s_t, a_t) = \mathbb{E}_{S_{t+1}, A_{t+1}}[R_t + \gamma \cdot Q_*(S_{t+1}, A_{t+1}) \big| S_t = s_t, A_t = a_t]
$$

因为动作 $A_{t+1} = \arg\max_A Q_*(S_{t+1}, A)$ 是状态 S_{t+1} 的确定性函数，所以

$$
Q_*(s_t, a_t) = \mathbb{E}_{S_{t+1}}[R_t + \gamma \cdot \max_{A \in \mathcal{A}} Q_*(S_{t+1}, A) \big| S_t = s_t, A_t = a_t]
$$

定理 A.4 最优贝尔曼方程：假设 R_t 是 S_t, A_t, S_{t+1} 的函数，那么

$$
Q_*(s_t, a_t) = \mathbb{E}_{S_{t+1} \sim p(\cdot | s_t, a_t)}[R_t + \gamma \cdot \max_{A \in \mathcal{A}} Q_*(S_{t+1}, A) \big| S_t = s_t, A_t = a_t]
$$