一、 基础知识

1.1 蒙特卡洛方法

1.1.1 近似阴影面积 A

如果我们有一个不等式，例如单位圆阴影面积的不等式 $x^2 + y^2 \leq 1$，要求出它的面积，即使使用定积分，也是比较复杂的。

这时候我们就可以使用蒙特卡洛方法，使用一个正方形将其包围住，并且在正方形内进行均匀随机抽样，然后根据约等式

$$\frac{\text{单位圆面积}}{\text{正方形面积}} \approx \frac{\text{满足不等式的抽样点}}{\text{所有抽样点}}$$

即可得到

$$\text{单位圆面积} \approx \text{正方形面积} \times \frac{\text{满足不等式的抽样点}}{\text{所有抽样点}}$$

1.1.2 近似定积分

为了计算集合 Ω 上的定积分 $I = \int_\Omega f(x)\,\mathrm{d}x$，我们进行下列的步骤:

1. 在集合 Ω 上做 **均匀随机抽样**，得到 n 个样本，记作 $x_1, \cdots, x_n$;
2. 计算集合 Ω 的面积

$$v = \int_\Omega \mathrm{d}x$$

3. 对函数值 $f(x_1), \cdots, f(x_n)$ 求平均，再乘以 Ω 面积 v:

$$q_n = v \cdot \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

4. 返回 q_n 作为定积分 I 的估计值。

如果我们认为 Ω 满足不等式约束，则其中的重点在于:

1. 在 Ω 上做均匀随机抽样，一种容易想到的方法是类似于上面说的抽样，并检查是否满足不等式，不满足则丢弃，重新抽样。但这种方法麻烦的地方在于需要找到一个比较好的「盒子」将其囊括起来，这点也并不是那么朴素的（类似于接受-拒绝抽样）。
2. 计算 Ω 的面积也并不是那么简单，不过这里也同样可以采用上文提到的计算阴影面积的蒙特卡洛方法。

1.1.3 近似期望

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

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

可以看出这是一个定积分，我们可以用上面说到的 **均匀抽样** 的方式求值，但是我们使用 **非均匀抽样** 会更快。

将原来的期望式子写成

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

则可知我们可以采用下面的计算方法

1. 按照概率密度函数 $p(x)$，在集合 Ω 上做非均匀随机抽样，得到 n 个样本 $x_1, \cdots, x_n \sim p(\cdot)$；
2. 对函数值 $f(x_1), \cdots, f(x_n)$ 求平均：

$$q_n = \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

3. 返回 q_n 作为期望 $\mathbb{E}_{p(\cdot)}[f(X)]$ 的估计值。

这里有两点要注意：

1. 如何按照概率密度函数 $p(x)$ 进行抽样？这就涉及到了机器学习中常用的抽样方法了。参考链接：https://www.cnblogs.com/vpegasus/p/sampling.html
   - **Inverse CDF**：最简单直观的方法，就是求分布函数（CDF）的反函数。这是因为如果有一个样本集是遵循 $p(x)$ 采样得到的，则其对应的累积分布函数值集合是也是随机的，且服从 $(0, 1)$ 上的均匀分布。所以我们只需要在 $(0, 1)$ 范围内随机均匀抽样，在通过分布函数的反函数即可求得概率密度函数 p 的抽样值。
     - 证明：令 $Z = F(X)$，其中 F 是 X 的分布函数，则有

       $$\mathbb{P}(Z \leq z) = P(F(X) \leq z) = P\big(X \leq F^{-1}(z)\big) = F\big(F^{-1}(z)\big) = z$$

       这是均匀分布的 CDF。
   - **接受-拒绝采样**（Acceptance-Rejection Sampling）：当对某一分布 $p(x)$ 直接抽样比较困难时，可以通过对另一相对容易的分布 $q(x)$ 进行抽样，然后保留其中服从 $p(x)$ 的样本，而剔除不服从 $p(x)$ 的无效样本。$q(x)$ 称为建议分布函数（proposal distribution），必须能够「罩住」待抽样分布 $p(x)$（可以乘以一个系数 k 达到这个目的）。
     a. 从 $q(x)$ 随机抽取一个样本 x_i；
     b. 从均匀分布 $U(0, 1)$ 中随机抽取一个样本 u_i；

c. 比较 u_i 与 $\alpha = \frac{p(x)}{k \cdot q(x)}$，如果 $u_i \leq \alpha$ 则认为 x_i 为服从 $p(x)$ 的有效样本，反之，则认为无效，丢弃。

- **蒙特卡洛方法**
  - 重要性采样：与接受-拒绝采样类似，都是引入一个相对容易采样的分布，不过没有接受与否的判定，而是对所有样本进行加权平均，想法非常直观

$$\int_{\Omega} p(x) \cdot f(x) \, \mathrm{d}x = \int_{\Omega} \frac{p(x)}{q(x)} \cdot q(x) \cdot f(x) \, \mathrm{d}x = \int_{\Omega} w(x) \cdot q(x) \, \mathrm{d}x$$

  - MCMC（Markov Chain Monte Carlo）
  - M-H 算法
  - Gibbs Sampling

2. 为什么这里不再需要算 Ω 面积？因为这里

$$\int_{\Omega} \mathrm{d}P(x) = \int_{\Omega} p(x) \, \mathrm{d}x = 1$$

1.2 马尔可夫决策过程（MDP）

1.2.1 基本概念

1. **状态**（State）：状态是对当前环境的一个概括，或者说是做决策的唯一依据；
2. **状态空间**（State Space）：所有可能存在状态的集合，记作花体字母 $\mathcal{S}$；
3. **动作**（Action）：做出的决策；
4. **动作空间**（Action Space）：所有可能动作的集合；
5. **智能体**（Agent）：做动作的主体；
6. **策略函数**（Policy Function）：根据观测到的状态做出决策，控制智能体动作；
7. **奖励**（Reward）：智能体执行一个动作后，环境返回给智能体的一个数值；
8. **状态转移函数**（State-Transition Function）：环境生成新状态 s' 时会用到的函数，记作 $p(s'|s,a) = \mathbb{P}(S' = s' \mid S = s, A = a)$。

1.2.2 随机性来源

- 强化学习中随机性来源有两个：策略函数和状态转移函数。
  - **动作** 的随机性来源于 **策略函数**；
  - **状态** 的随机性来源于 **状态转移函数**；
- **奖励** 可以看作状态和动作的函数。
- **轨迹** 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。

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

1.2.3 回报

- **回报**（Return）是从当前时刻开始到一回合结束的所有奖励的综合，所以回报也叫 **累积奖励**。
- **折扣回报（Dicounted Return）** 定义：$U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

1.2.4 价值函数

1.2.4.1 动作价值函数

我们需要回报来衡量局势的好坏，但是回报是一个随机变量，怎么办呢？解决方案就是对 U_t 求期望，消除掉其中的随机性。

假设我们已经观测到状态 s_t，并且做完决策，选中动作 a_t。那么对

$$S_{t+1}, A_{t+1}, \quad S_{t+1}, A_{t+1}, \quad \cdots, \quad S_{t+1}, A_{t+1}$$

求条件期望，就得到 **动作价值函数**（Action-Value Function）：

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

可以看出动作价值函数 $Q_\pi(s_t, a_t)$ 依赖于 s_t、a_t 与策略函数 $\pi(a|s)$，而不依赖于 $t+1$ 时刻及之后的状态和动作，因为后者被期望消除了。

1.2.4.2 最优价值函数

如何排除 π 的影响，指评价当前状态和动作的好坏呢？解决方案就是 **最优动作价值函数**（Optimal Action-Value Function）：

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

最优策略函数：

$$\pi^\star(s_t, a_t) = \operatorname*{argmax}_\pi Q_\pi(s_t, a_t), \quad \forall s_t \in \mathcal{S}, a_t \in \mathcal{A}$$

1.2.4.3 状态价值函数

状态价值函数不依赖于动作：

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

状态价值函数 $V_\pi(s_t)$ 也是回报 U_t 的期望：

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

用状态价值函数可以衡量策略 π 和状态 s_t 的好坏。

二、 强化学习分类

- 强化学习方法
  - **基于模型的方法**
  - **无模型方法**
    - **价值学习**（Value-Based Learning）通常是指学习最优价值函数 $Q_\star(s,a)$（或动作价值函数、状态价值函数）
      - **深度 Q 网络**（DQN）：最有名的价值学习方法，用一个神经网络近似 $Q_\star$
        - 使用 TD 算法进行训练
          - 异策略 TD 算法：Q 学习，用于近似 $Q_\star$，可以使用经验回放
          - 同策略 TD 算法：SARSA，用于近似 Q_π，不能使用经验回放
      - 用表格表示 $Q_\star$
    - 策略学习（Policy-Based Learning）指的是学习策略函数 $\pi(a \mid s)$。
      - 策略梯度等方法

三、 价值学习

3.1 DQN 与 Q 学习

DQN 记作 $Q(s,a;w)$，是用于近似「先知」最优动作值函数 $Q_\star$ 的神经网络。

DQN 的梯度：在训练 DQN 时，需要对 DQN 关于神经网络参数 w 求梯度。用

$$\nabla_w Q(s,a;w) \triangleq \frac{\partial Q(s,a;w)}{\partial w}$$

表示函数值 $Q(s,a;w)$ 关于参数 w 的梯度，形状与 w 完全相同。

3.2 时间差分（TD）算法

设有一条路径 $s \to m \to d$，我们要预测路径耗时。

- 原始预测：$\hat{q} \triangleq Q(s,d;w)$
- 实际花费：y
- TD 目标（TD Target）：$\hat{y} \triangleq r + \hat{q}'$
- 损失函数：$L(w) = \frac{1}{2}(\hat{q} - \hat{y})^2$
- 损失函数梯度：$\nabla_w L(w) = (\hat{q} - \hat{y}) \cdot \nabla_w Q(s,d;w)$
  - 此处将 $\hat{y}$ 看做常数
- TD 误差（TD Error）：$\delta = \hat{q} - \hat{y}$
- 梯度下降更新模型：$w \leftarrow w - \alpha \cdot \delta \cdot \nabla_w Q(s,d;w)$

3.3 用 TD 训练 DQN

3.3.1 Q 学习算法推导

由回报的定义 $U_t = \sum_{k=t}^{n} \gamma^{k-t} \cdot R_k$ 可得最优贝尔曼方程

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

贝尔曼方程右侧是一个期望，可以做蒙特卡洛近似。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}，也就有奖励 r_t，即有四元组

$$(s_t, a_t, r_t, s_{t+1})$$

计算出

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

则可以看作 $Q_\star(s_t, a_t)$ 的蒙特卡洛近似，即有

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

替换成神经网络得到

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

使用均方误差损失函数并做梯度下降可得训练 DQN 的 TD 算法的公式

$$w \leftarrow w - \alpha \cdot \delta_t \cdot \nabla_w Q(s_t, a_t; w)$$

其中 $\delta_t = \hat{q}_t - \hat{y}_t$ 是 TD 误差 δ_t。

3.3.2 用 Q 学习训练 DQN

1. **收集训练数据**：使用行为策略（Behavior Prolicy）如 ε-greedy 策略

$$a_t = \begin{cases} \text{argmax}_a \, Q(s_t, a; w), & \text{以概率 } (1-\varepsilon) \\ \text{均匀抽取 } \mathcal{A} \text{ 中的一个动作}, & \text{以概率 } \varepsilon \end{cases}$$

   收集到一局游戏中的轨迹，并且划分成 n 个 (s_t, a_t, r_t, s_{t+1}) 这种四元组，存入 **经验回放数组**（Replay Buffer）。
2. **更新 DQN 参数 w**：取出一个四元组 (s_j, a_j, r_j, s_{j+1})。
   a. 对 DQN 做正向传播，得到 Q 值：

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

   b. 计算 TD 目标和 TD 误差：

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

   c. 对 DQN 做反向传播，得到梯度：

$$g_j = \nabla_w Q(s_j, a_j; w_{\text{now}})$$

   d. 做梯度下降更新 DQN 的参数：

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

同样地，我们也可以用 Q 学习来训练用表格表示的 $Q_\star$。

3.3.3 同策略和异策略

- **行为策略**：训练时用于收集经验的策略；

- **目标策略**：在结束后得到的用于控制智能体行动的策略；
- **同策略（On-policy）**：行为策略和目标策略 **必须** 是相同的；
- **异策略（Off-policy）**：行为策略和目标策略 **可以** 是不同的；
- **异策略优势**：可以使用行为策略收集经验，并存放在经验回放数组中，用于反复更新目标策略。同策略就不能使用经验回放数组，因为经验回放数组中的 **旧数据** 对应的是旧行为策略，与不断实时更新的目标策略所需的经验不一致。

3.3.4 SARSA 学习算法推导

SARSA 也是一种 TD 算法，其目标与 Q 学习不同，SARSA 的目的是学习动作价值函数 $Q_\pi(s,a)$。

为什么要学习 $Q_\pi(s,a)$ 而不是像 Q 学习一样学习 $Q_\star(s,a)$？这里学习得到 Q_π 并不是用于控制智能体，而是用来评价策略函数 π 的好坏。这被称为 Actor-Critic（演员-评委）方法，这是一种策略学习算法，策略函数 π 控制智能体，被看作「演员」；而 Q_π 评价 π 的表现，用于帮忙改进 π，被看作「评委」。Actor-Critic 通常用 SARSA 训练「评委」Q_π。

我们有贝尔曼方程：

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

因此我们可以对贝尔曼方程两边做近似：

- 方程左边 $Q_\pi(s_t,a_t)$ 近似成 $q(s_t,a_t)$。
- 方程右边是对下一时刻状态 S_{t+1} 和动作 A_{t+1} 求的期望。给定 s_t,a_t，执行状态转移函数得到奖励 r_t 和新状态 s_{t+1}，然后基于 s_{t+1} 做随机抽样，得到新动作

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

用观测得到的 r_t、s_{t+1} 和计算出的 $\tilde{a}_{t+1}$ 对期望做蒙特卡洛近似，并将 Q_π 近似成 q，得到

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

它即使 TD 目标。

SARSA 是 State-Action-Reward-State-Action 的缩写，原因是 SARSA 算法用到了这个五元组：$(s_t,a_t,r_t,s_{t+1},\tilde{a}_{t+1})$。SARSA 算法学到的 q 依赖于策略 π，这是因为五元组中的 $\tilde{a}_{t+1}$ 是根据 $\pi(\cdot\mid s_{t+1})$ 抽样得到的。这也是 SARSA 是同策略算法且无法使用经验回放的原因，即策略 π 是实时更新的，不能使用过时的经验数据。

3.3.5 神经网络形式的 SARSA

训练流程：设当前价值网络参数为 w_{now}，当前策略为 π_{now}。每一轮训练用五元组 $(s_t,a_t,r_t,s_{t+1},\tilde{a}_{t+1})$ 对价值网络参数进行一次更新。

1. 观测到当前状态 s_t，根据当前策略做抽样：$a_t\sim\pi_{\text{now}}(\cdot\mid s_t)$。
2. 用价值网络计算 (s_t,a_t) 的价值：

$$\hat{q}_t = q(s_t, a_t; w_{\text{now}})$$

3. 智能体执行动作 a_t 之后，观测到奖励了 r_t 和新的状态 s_{t+1}。
4. 根据当前策略做抽样：$\tilde{a}_{t+1} \sim \pi_{\text{now}}(\cdot \mid s_{t+1})$。注意，$\tilde{a}_{t+1}$ 只是假想的动作，智能体不执行。
5. 用价值网络计算 $(s_{t+1}, \tilde{a}_{t+1})$ 的价值：

$$\hat{q}_{t+1} = q(s_{t+1}, \tilde{a}_{t+1}; w_{\text{now}})$$

6. 计算 TD 目标和 TD 误差：

$$\hat{y}_t = r_t + \gamma \cdot \hat{q}_{t+1}, \quad \delta_t = \hat{q}_t - \hat{y}_t$$

7. 对价值网络 q 做反向传播，计算 q 关于 w 的梯度：$\nabla_w q(s_t, a_t; w_{\text{now}})$。
8. 更新价值网络参数：

$$w_{\text{new}} \leftarrow w_{\text{now}} - \alpha \cdot \delta_t \cdot \nabla_w q(s_t, a_t; w_{\text{now}})$$

9. 用某种算法更新策略函数（策略学习）。该算法与 SARSA 算法无关。

3.3.6 多步 TD 目标的 SARSA

训练流程：设当前价值网络参数为 w_{now}，当前策略为 π_{now}。执行以下步骤更新价值网络和策略。

1. 用策略网络 π_{now} 控制智能体与环境交互，完成一个回合，得到轨迹

$$s_1, a_1, r_1, \quad s_2, a_2, r_2, \quad \cdots, \quad s_n, a_n, r_n$$

2. 对于所有 $t = 1, \cdots, n-m$，计算

$$\hat{q}_t = q(s_t, a_t; w_{\text{now}})$$

3. 对于所有的 $t = 1, \cdots, n-m$，计算多部 TD 目标和 TD 误差：

$$\hat{y}_t = \sum_{i=0}^{m-1} \gamma^i r_{t+i} + \gamma^m \hat{q}_{t+m}$$

4. 对于所有的 $t = 1, \cdots, n-m$，对价值网络 q 做反向传播，计算 q 关于 w 的梯度：

$$\nabla_w q(s_t, a_t; w_{\text{now}})$$

5. 更新价值网络参数：

$$w_{\text{new}}(s_t, a_t) \leftarrow w_{\text{now}}(s_t, a_t) - \alpha \cdot \sum_{t=1}^{n-m} \delta_t \cdot \nabla_w q(s_t, a_t; w_{\text{now}})$$

6. 用某种算法更新策略函数 π。该算法与 SARSA 无关。

3.3.7 蒙特卡洛与自举

- **蒙特卡洛**：多步 TD 目标如果直接将一局游戏进行到底再进行计算，这就是已经不是 TD 了，而是直接被成为「蒙特卡洛」。
  - 优点是 **无偏性**：u_t 是 $Q_\pi(s_t, a_t)$ 的无偏估计。
  - 缺点是 **方差大**：随机变量多，不确定大，因此拿 u_t 作为目标训练价值网络，收敛会很慢。

- **自举**：用一个估算去更新同类的估算，类似于「自己把自己举起来」，在这里就是多步 TD 目标，以单步 TD 目标的自举程度最大。
  - 优点是 **方差小**。
  - 缺点是 **有偏差**，自举会导致偏差传播。
- 如果设置一个较好的 m，则能在方差和偏差之间找到较好的平衡。

3.4 高级技巧

3.4.1 经验回放

- 异策略算法（如 Q 学习）可以使用经验回放数组。
- 实践中，要等回放数组中有足够多四元组时，才开始做经验回放更新 DQN。
- 经验回放优点
  - **打破序列相关性**：我们希望相邻两次使用的四元组是独立的，而收集经验时两个时间上相邻收集的四元组有很的相关性，因此我们要从数组里随机抽取，这样消除了相关性。
  - **重复利用收集到的经验**：可以使用更少的样本达到同样的表现。
- 局限性
  - **不能用于同策略算法**，如 SARSA

3.4.2 优先经验回放

- 部分样本可能比其他样本更稀少却也更重要，如无人车碰到像旁边车辆强行变道等意外情况。
- 可以给每一个四元组一个权重，根据权重做非均匀随机抽样。
- 自动判断哪些样本更重要：**TD 误差大的样本被抽到的概率应该更大**
  - $p_j \propto |\delta_j| + \varepsilon$
  - $p_j \propto \frac{1}{\mathrm{rank}(j)}$
- 还需要将权重大的样本学习率调小（用大学习率更新一次梯度、用小学习率更新多次梯度并不等价，第二种方式是对样本更有效的利用）
  - $\alpha_j = \frac{\alpha}{(b \cdot p_j)^\beta}$
  - 论文中建议一开始 β 比较小，最终增长到 1

3.4.3 高估问题

Q 学习算法有一个缺陷：用 Q 学习训练出的 DQN 会高估真实的价值，且高估通常是非均匀的。原因有两个：

1. 自举导致偏差的传播。
2. 最大化导致高估。
   - 由于有噪声的存在，可以将神经网络写成 $Q(s,a;w) = Q_\star(s,a) + \varepsilon$
   - 因此有不等式 $\mathbb{E}_\varepsilon[\max_{a\in\mathcal{A}} Q(s,a;w)] \geq \max_{a\in\mathcal{A}} Q_\star(s,a)$

缓解高估问题：

1. 目标网络：切断自举的传播。
2. 双 Q 学习算法：缓解最大化造成的高估。

	选择	求值	自举造成偏差	最大化造成高估
原始 Q 学习	DQN	DQN	严重	严重
Q 学习 + 目标网络	目标网络	目标网络	不严重	严重
双 Q 学习	DQN	目标网络	不严重	不严重

表 1: 三种 TD 算法的对比

SARSA 算法不存在最大化这个问题，但仍然存在自举问题，因此应该使用目标网络。

3.4.4 噪声网络

噪声网络（Noisy Net）是一种非常简单的方法，可以显著提高 DQN 的表现。噪声网络应用不局限于 DQN，它可以用于几乎所有的强化学习方法。

噪声网络的原理即把神经网络中的参数 w 替换成 $\mu + \sigma \circ \xi$，μ、σ 分别是均值和方差，而 ξ 是从标准正态分布中抽样的随机噪声。

噪声网络训练出来的 DQN 有稳健性：参数不严格等于 μ 也没关系，只要在参数 μ 的领域内，做出的预测都比较合理，不会「失之毫厘，谬以千里」。

四、 策略学习

策略学习的意思是通过求解一个优化问题，学出最优策略函数或它的近似。

4.1 策略网络

对于离散动作空间来说，策略函数 π 是个条件概率质量函数：

$$\pi(a|s) \triangleq \mathbb{P}(A = a \mid S = s)$$

为了得到这样要给策略函数，当前最有效的方法是通过神经网络 $\pi(a|s;\theta)$ 近似策略函数 $\pi(a|s)$。神经网络 $\pi(a|s;\theta)$ 被称为策略网络。

4.2 策略学习的目标函数

我们有动作值函数

$$Q_\pi(s_t, a_t) = \mathbb{E}[U_t \mid S_t = s_t, A_t = a_t]$$

则也有状态值函数

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

如果一个策略很好，那么对所有的状态 S，状态价值 $V_\pi(S)$ 的均值应该很大，因此我们可以定义目标函数

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

这个目标函数排除了状态 S 的影响，只依赖于策略网络 π 的参数 θ，因此策略学习可以描述为优化问题

$$\max_\theta J(\theta)$$

我们使用梯度上升可得

$$\theta_{\text{new}} \leftarrow \theta_{\text{now}} + \beta \cdot \nabla_\theta J(\theta_{\text{now}})$$

其中梯度

$$\nabla_\theta J(\theta_{\text{now}}) \triangleq \frac{\partial J(\theta)}{\partial \theta}\Big|_{\theta=\theta_{\text{now}}}$$

被称为策略梯度，我们可以使用 **策略梯度定理**：

策略梯度定理（不严谨表述）

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

更严格的表述是

策略梯度定理（完整表述）

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1-\gamma^n}{1-\gamma}\mathbb{E}_{S\sim d(\cdot)}\left[\mathbb{E}_{A\sim\pi(\cdot|S;\theta)}\left[\frac{\partial \ln \pi(A|S;\theta)}{\partial \theta}\cdot Q_\pi(S,A)\right]\right]$$

其中 $d(\cdot)$ 是马尔科夫链的稳态分布，而系数 $\frac{1-\gamma^n}{1-\gamma}$ 无关紧要，因为会被学习率 β 吸收掉。

4.3 近似策略梯度

为了进行梯度上升：

$$\theta_{\text{new}} \leftarrow \theta_{\text{now}} + \beta \cdot \nabla_\theta J(\theta_{\text{now}})$$

策略梯度定理 证明：

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

解析求出这个期望是不可能的，因为我们并不知道状态 S 概率密度函数，即使我们知道，我们也不愿意这样做，因为连加或定积分的计算量非常大。

我们可以使用期望的蒙特卡洛近似，用于近似策略梯度中的期望。每次从环境中观测一个状态 s，它相当于随机变量 S 的观测值，然后再根据当前最新的策略网络随机抽样得出一个动作：

$$a \sim \pi(\cdot\,|s;\theta)$$

计算得到随机梯度

$$g(s,a;\theta) \triangleq Q_\pi(s,a)\cdot\nabla_\theta \ln \pi(a|s;\theta)$$

很显然，$g(s,a;\theta)$ 是策略梯度 $\nabla_\theta J(\theta)$ 的无偏估计，于是我们有结论

结论

随机梯度 $g(s,a;\theta) \triangleq Q_\pi(s,a)\cdot\nabla_\theta \ln \pi(a|s;\theta)$ 是策略梯度 $\nabla_\theta J(\theta)$ 的无偏估计。

应用上述结论，则可以通过随机梯度上升来更新 θ：

$$\theta \leftarrow \theta + \beta \cdot g(s, a; \theta)$$

但是这种方法仍然不可行，因为我们不知道动作价值函数 $Q_\pi(s, a)$。在后面，我们可以有两种方法对 $Q_\pi(s, a)$ 近似，一种是 REINFORCE，用实际观测的回报 u 近似 $Q_\pi(s, a)$；另一种方法是 Actor-Critic，用神经网络 $q(s, a; w)$ 近似 $Q_\pi(s, a)$。

4.4 REINFORCE

由于动作价值定义为 U_t 的条件期望

$$Q_\pi(s_t, a_t) = \mathbb{E}[U_t \mid S_t = s_t, A_t = a_t]$$

让智能体完成一局游戏，观测到所有奖励，然后计算出 $u_t = \sum_{k=t}^{n} \gamma^{k-t} \cdot r_k$。由于 u_t 是 U_t 的观测值，所以是该公式的蒙特卡洛近似。实践中，可以使用 u_t 代替 $Q_\pi(s_t, a_t)$，那么随机梯度 $g(s_t, a_t; \theta)$ 可以近似成

$$\tilde{g}(s_t, a_t; \theta) = u_t \cdot \nabla_\theta \ln \pi(a_t | s_t; \theta)$$

$\tilde{g}$ 是 g 的无偏估计，所以也是策略梯度 $\nabla_\theta J(\theta)$ 的无偏估计；$\tilde{g}$ 也是一种随机梯度。

这种方法就叫 REINFORCE。

训练流程：

1. 用策略网络 θ_{now} 控制智能体从头玩一局游戏，得到一条轨迹：

$$s_1, a_1, r_1, \quad , s_2, a_2, r_2, \quad \cdots, \quad , s_n, a_n, r_n$$

2. 计算所有的回报：

$$u_t = \sum_{k=t}^{n} \gamma^{k-t} \cdot r_k, \quad \forall t = 1, \cdots, n$$

3. 用 $\{(s_t, a_t)\}_{t=1}^{n}$ 作为数据，做反向传播计算：

$$\nabla_\theta \ln \pi(a_t | s_t; \theta_{\text{now}}), \quad \forall t = 1, \cdots, n$$

4. 做随机梯度上升更新策略网络参数：

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

注意，REINFORCE 是一种 **同策略** 方法，要求行为策略与目标策略相同，因此不适用经验回放。

4.5 Actor-Critic

Actor-Critic 方法使用神经网络近似动作价值函数 $Q_\pi(s, a)$，这个神经网络叫做价值网络，记作 $q(s, a; w)$。

价值网络与 DQN 的区别：

- 价值网络是对动作价值函数 $Q_\pi(s,a)$ 的近似，而 DQN 则是对最优动作价值函数 $Q_\star(s,a)$ 的近似。
- 对价值网络的训练使用的是 SARSA 算法，是同策略算法，不能用经验回放；而 DQN 训练使用的是 Q 学习算法，属于异策略，可以用经验回放。

Actor-Critic 翻译成「演员–评委」方法。策略网络 $\pi(a|s;\theta)$ 相当于演员，它基于状态 s 做出动作 a。价值网络 $q(s,a;w)$ 相当于评委，它给演员打分，量化在状态 s 下做动作 a 的好坏程度。

为什么我们不能直接将当前奖励 R 反馈给策略网络（演员），而是要用价值网络（评委）这一个中介呢？这是因为策略学习的目标函数 $J(\theta)$ 是回报 U 的期望，而不是奖励 R 的期望。虽然我们能观测到当前奖励 R，但是它对策略网络毫无意义。而我们使用 R 帮忙训练价值网络，价值网络经过训练后能够估算出回报 U 的期望。

而加入中介价值网络（评委）的好处是不再需要像 REINFORCE 一样要完成一整局游戏后才能进行一次更新。

将之前的策略梯度无偏估计：

$$g(s,a;\theta) \triangleq Q_\pi(s,a)\cdot\nabla_\theta \ln \pi(a|s;\theta)$$

里的动作价值函数 $Q_\pi(s,a)$ 替换成价值网络即可得到近似策略梯度：

$$\hat{g}(s,a;\theta) \triangleq q(s,a;w)\cdot\nabla_\theta \ln \pi(a|s;\theta)$$

最后做梯度上升即可。

训练流程（使用 SARSA 训练，含目标网络）：

1. 观测到当前状态 s_t，根据策略网络做决策：$a_t \sim \pi(\cdot\,|s_t;\theta_{\text{now}})$，并让智能体执行动作 a_t。
2. 从环境中观测到奖励 r_t 和新状态 s_{t+1}。
3. 根据策略网络做决策：$\tilde{a}_{t+1} \sim \pi(\cdot\,|s_{t+1};\theta_{\text{now}})$，但不让智能体执行动作 $\tilde{a}_{t+1}$。
4. 让价值网络给 s_t, a_t 打分：

$$\hat{q}_t = q(s_t,a_t;w_{\text{now}}).$$

5. 让目标网络给 $s_{t+1}, \tilde{a}_{t+1}$ 打分：

$$\hat{q}^-_{t+1} = q(s_{t+1},\tilde{a}_{t+1};w^-_{\text{now}}).$$

6. 计算 TD 目标与 TD 误差：

$$\hat{y}^-_t = r_t + \gamma\cdot\hat{q}^-_{t+1} \quad \text{和} \quad \delta_t = \hat{q}_t - \hat{y}^-_t.$$

7. 更新价值网络：

$$w_{\text{new}} \leftarrow w_{\text{now}} - \alpha \cdot \delta_t \cdot \nabla_w q(s_t, a_t; w_{\text{now}}).$$

8. 更新策略网络：

$$\theta_{\text{new}} \leftarrow \theta_{\text{now}} + \beta \cdot \widehat{q}_t \cdot \nabla_\theta \ln \pi(a_t|s_t; \theta_{\text{now}}).$$

9. 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：

$$w^-_{\text{new}} \leftarrow \tau \cdot w_{\text{new}} + (1 - \tau) \cdot w^-_{\text{now}}.$$

五、 带基线的策略梯度方法

带基线的策略梯度（Policy Gradient with Baseline）可以大幅提升策略梯度方法的表现。使用基线后，REINFORCE 变成 REINFORCE with Baseline，Actor-Critic 变成 Advantage Actor-Critic（A2C）。

5.1 策略梯度中的基线

策略梯度定理 证明：

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

我们只需要做一个小改动，就能大幅提升表现：把 b 作为动作价值函数 $Q_\pi(S,A)$ 的基线（Baseline），用 $Q_\pi(S,A)-b$ 替换掉 Q_π。设 b 是任意的函数，只要不依赖于动作 A 即可；例如 b 可以是状态价值函数 $V_\pi(S)$。

带基线的策略梯度定理

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

其原因在于

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

证明

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

使用蒙特卡洛近似，从环境中观测一个状态 s，然后根据策略网络抽样得到 $a \sim \pi(\cdot\,|s;\theta)$。那么策略梯度 $\nabla_\theta J(\theta)$ 可以近似为下面的随机梯度：

$$g_{b(s,a;\theta)} = [Q_\pi(s,a) - b] \cdot \nabla_\theta \ln \pi(a|s;\theta)$$

得到的随机梯度 $g_{b(s,a;\theta)}$ 是 $\nabla_\theta J(\theta)$ 的无偏估计：

$$\text{Bias} = \mathbb{E}_{S,A}\left[g_{b(S,A;\theta)}\right] - \nabla_\theta J(\theta) = 0$$

但是 b 的取值会对方差有影响，方差为

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

如果 b 很接近 $Q_\pi(s,a)$ 关于 a 的均值，那么方差会比较小，所以 $b = V_\pi(s)$ 是很好的基线。

基线解释

为什么我们能减去基线呢？这是因为我们在意的是动作价值函数各个动作对应值的相对大小，如果都减去基线，相对大小是不变的，这是我们能够使用基线的原因。

至于为什么使用了基线能提高策略梯度算法效果？我还得再看看。

5.2 带基线的 REINFORCE 算法

REINFORCE 使用实际观测的回报 u 来代替动作价值 $Q_\pi(s,a)$。为了使用基线，我们还需要一个神经网络 $v(s;w)$ 近似状态价值函数 $V_\pi(s)$。这样一来，$g(s,a;\theta)$ 就被近似成了：

$$\tilde{g}(s,a;\theta) = [u - v(s;w)] \cdot \nabla_\theta \ln \pi(a|s;\theta)$$

5.2.1 策略网络和价值网络

带基线的 REINFORCE 需要两个神经网络：策略网络 $\pi(a|s;\theta)$ 和价值网络 $v(s;w)$；$v(s;w)$ 的输入是状态 s，输出是一个实数。策略网络和价值网络输入都是状态 s，因此可以让两个神经网络共享 **卷积网络** 的参数，这是编程实现中常用的技巧。

注意，这里的价值网络并不是用作评委，这里与 A2C 是不同的。

5.2.2 训练流程

1. 用策略网络 θ_{now} 控制智能体从头玩一局游戏，得到一条轨迹：

$$s_1, a_1, r_1, \quad , s_2, a_2, r_2, \quad \cdots, \quad , s_n, a_n, r_n$$

2. 计算所有的回报：

$$u_t = \sum_{k=t}^{n} \gamma^{k-t} \cdot r_k, \quad \forall t = 1, \cdots, n$$

3. 让价值网络做预测：

$$\hat{v}_t = v(s_t; w_{\text{now}}), \quad \forall t = 1, \cdots, n.$$

4. 计算误差 $\delta_t = \hat{v}_t - u_t, \quad \forall t = 1, \cdots, n$。
5. 用 $\{s_t\}_{t=1}^{n}$ 作为价值网络输入，做反向传播计算：

$$\nabla_w v(s_t; w_{\text{now}}), \quad \forall t = 1, \cdots, n$$

6. 用 $\{(s_t, a_t)\}_{t=1}^{n}$ 作为数据，做反向传播计算：

$$\nabla_\theta \ln \pi(a_t|s_t; \theta_{\text{now}}), \quad \forall t = 1, \cdots, n$$

7. 做随机梯度上升更新策略网络参数：

$$\theta_{\text{new}} \leftarrow \theta_{\text{now}} - \beta \cdot \sum_{t=1}^{n} \gamma^{t-1} \cdot \underbrace{\delta_t \cdot \nabla_\theta \ln \pi(a_t|s_t; \theta_{\text{now}})}_{\text{负的近似梯度 } -\tilde{g}(s_t, a_t; \theta_{\text{now}})}$$

5.3 Advantage Actor-Critic（A2C）

我们之前得到的策略梯度的无偏估计

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

公式中的 $Q_\pi - V_\pi$ 被称为优势函数（Advantage Function）。因此，基于上面公式得到的 Actor-Critic 方法被称为 Advantage Actor-Critic，缩写 A2C。

这种方法也有一个价值网络 $v(s; w)$，在这里是作为评委，他的评分可以帮助策略网络（演员）改进技术。这里的神经网络结构与上一节相同，但是训练方法不同。

可以注意到，这里也和不带基线的 Actor-Critic 方法不同，这里没有用到动作价值网络 $q(s, a; w)$，而是使用了价值网络 $v(s; w)$，而前者可以通过贝尔曼公式得到。

5.3.1 算法推导

训练价值网络：

根据贝尔曼公式：

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

我们对两边做近似：

- 方程左边的 $V_\pi(s_t)$ 近似成 $\hat{v}_t \triangleq v(s_t; w)$，这是价值网络在 t 时刻对 $V_\pi(s_t)$ 做出的估计。。

- 方程右边的期望，给定 s_t，智能体执行动作 a_t，环境给出奖励 r_t 和新状态 s_{t+1}，然后蒙特卡洛近似得到 TD 目标：

$$\widehat{y}_t \triangleq r_t + \gamma \cdot v(s_{t+1}; w)$$

  这是价值网络在 $t+1$ 时刻对 $V_\pi(s_t)$ 做出的估计。

之后便是照常地定义均方损失函数与求梯度，然后做梯度下降。

训练策略网络：

为了对 $g(s, a; \theta)$ 做近似，我们继续使用贝尔曼公式：

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

因此使用蒙特卡洛近似可得

$$\tilde{g}(s_t, a_t; \theta) \triangleq \left[\underbrace{r_t + \gamma \cdot v(s_{t+1}; w)}_{\text{TD 误差 } \widehat{y}_t} - v(s_t; w)\right] \cdot \nabla_\theta \ln \pi(a_t|s_t; \theta)$$

再根据 TD 目标和 TD 误差：

$$\widehat{y}_t \triangleq r_t + \gamma \cdot v(s_{t+1}; w) \quad \text{和} \quad \delta_t \triangleq v(s_t; w) - \widehat{y}_t$$

可得

$$\tilde{g}(s_t, a_t; \theta) \triangleq -\delta_t \cdot \nabla_\theta \ln \pi(a_t|s_t; \theta)$$

为什么价值网络不显示包含 a_t 也可以帮助策略网络（演员）改进演技?

价值网络 v 告诉策略网络 π 的唯一信息是 δ_t，根据其定义

$$-\delta_t = \underbrace{r_t + \gamma \cdot v(s_{t+1}; w)}_{\text{TD 目标 } \hat{\gamma}_t} - \underbrace{v(s_t; w)}_{\text{基线}}$$

有基线 $v(s_t; w)$ 是价值网络在 t 时刻对 $\mathbb{E}[U_t]$ 的估计，此时未执行动作 a_t；其 TD 目标 $\hat{y}_t$ 是价值网络在 $t+1$ 时刻对 $\mathbb{E}[U_t]$ 的估计，此时已执行动作 a_t。

- 如果 $\hat{y}_t > v(s_t; w)$, 说明动作 a_t 很好, 奖励 r_t 超出预期, 这时候应该更新 θ 使 $\pi(a_t|s_t; \theta)$ 变大。
- 如果 $\hat{y}_t < v(s_t; w)$, 说明动作 a_t 不好, 奖励 r_t 不及预期, 这时候应该更新 θ 使 $\pi(a_t|s_t; \theta)$ 减小。

这说明 δ_t 可以间接反映 a_t 的好坏。

5.3.2 训练流程（含目标网络）

1. 观测到当前状态 s_t，根据策略网络做决策：$a_t \sim \pi(\cdot\,|s_t; \theta_{\text{now}})$，并让智能体执行动作 a_t。
2. 从环境中观测到奖励 r_t 和新状态 s_{t+1}。
3. 让价值网络给 s_t 打分：

$$\hat{v}_t = v(s_t; w_{\text{now}}).$$

4. 让目标网络给 s_{t+1} 打分：

$$\hat{v}_{t+1}^{-} = v(s_{t+1}; w_{\text{now}}^{-}).$$

5. 计算 TD 目标与 TD 误差：

$$\hat{y}_t^{-} = r_t + \gamma \cdot \hat{v}_{t+1}^{-} \quad \text{和} \quad \delta_t = \hat{v}_t - \hat{y}_t^{-}.$$

6. 更新价值网络：

$$w_{\text{new}} \leftarrow w_{\text{now}} - \alpha \cdot \delta_t \cdot \nabla_w v(s_t; w_{\text{now}}).$$

7. 更新策略网络：

$$\theta_{\text{new}} \leftarrow \theta_{\text{now}} - \beta \cdot \delta_t \cdot \nabla_\theta \ln \pi(a_t|s_t; \theta_{\text{now}}).$$

8. 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：

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

六、 策略学习高级技巧

6.1 Trust Region Policy Optimization（TRPO）

置信域策略优化是一种策略学习方法，可以代替策略梯度方法。跟策略梯度方法相比，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 用更少的经验就能达到与策略梯度方法相同的表现。

学习 TRPO 的关键在于理解置信域方法（Trust Region Methods）。

6.1.1 置信域方法

置信域方法需要构造一个函数 $L(\theta|\theta_{\text{now}})$，这个函数要满足条件：

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

那么集合 $\mathcal{N}(\theta_{\text{now}})$ 就被称作 **置信域**。顾名思义，在 θ_{now} 的邻域上，我们可以信任 $L(\theta|\theta_{\text{now}})$，可以拿 $L(\theta|\theta_{\text{now}})$ 来替代目标函数 $J(\theta)$。

因此置信域方法主要分两步：

- **第一步——做近似**：给定 θ_{now}，构造函数 $L(\theta|\theta_{\text{now}})$，使得对于所有 $\theta \in \mathcal{N}(\theta_{\text{now}})$，函数值 $L(\theta|\theta_{\text{now}})$ 与 $J(\theta)$ 足够接近。
  - 近似的方法多种多样，如蒙特卡洛、二阶泰勒展开等。
- **第二步——最大化**：在置信域 $\mathcal{N}(\theta_{\text{now}})$ 中寻找变量 θ 的值，使得函数 L 的值最大化。把找到的值记作

$$\theta_{\text{new}} - \underset{\theta \in \mathcal{N}(\theta_{\text{now}})}{\operatorname{argmax}} L(\theta \mid \theta_{\text{now}})$$

  - 需要求解一个带约束的最大化问题，求解这个问题的数值优化方法很多，如梯度投影算法、拉格朗日法等。
  - 置信域 $\mathcal{N}(\theta_{\text{now}})$ 也有很多选择，既可以是球，也可以是两个概率分布的 KL 散度（KL Divergence）。

6.1.2 策略学习

状态价值函数可以转化为等价形式：

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_{A\sim\pi(\cdot|s;\theta)}[Q_\pi(s,A)] = \sum_{a\in\mathcal{A}} \pi(a|s;\theta)\cdot Q_\pi(s,a) \\ &= \sum_{a\in\mathcal{A}} \pi(a|s;\theta_{\text{now}})\cdot \frac{\pi(a|s;\theta)}{\pi(a|s;\theta_{\text{now}})}\cdot Q_\pi(s,a) \\ &= \mathbb{E}_{A\sim\pi(\cdot|s;\theta_{\text{now}})}\left[\frac{\pi(A|s;\theta)}{\pi(A|s;\theta_{\text{now}})}\cdot Q_\pi(s,A)\right] \end{aligned}$$

因此策略学习的目标函数 $J(\theta) = \mathbb{E}_S[V_\pi(S)]$ 也可以转化为等价形式。

目标函数等价形式

目标函数 $J(\theta)$ 可以等价写成：

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

上面 Q_π 中的 π 指的是 $\pi(A|S;\theta)$。

6.1.3 训练流程

1. 做近似——构造函数 $\tilde{L}$ 近似目标函数 $J(\theta)$：
   a. 设当前策略网络参数是 θ_{now}。用策略网络 $\pi(a|s;\theta_{\text{now}})$ 控制智能体与环境交互，玩完一局游戏，记下轨迹：

   $$s_1, a_1, r_1, \quad s_2, a_2, r_2, \quad \cdots, \quad s_n, a_n, r_n.$$

   b. 对于所有的 $t = 1, \cdots, n$，计算折扣回报 $u_t = \sum_{k=t}^{n} \gamma^{k-t}\cdot r_k$。
   c. 得出无偏的蒙特卡洛近似函数：

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

2. 最大化——用某种数值算法求解带约束的最大化问题：

   $$\theta_{\text{new}} = \underset{\theta}{\text{argmax}}\, \tilde{L}(\theta|\theta_{\text{now}}); \quad \text{s.t. } \|\theta - \theta_{\text{now}}\|_2 \le \Delta.$$

   此处的约束时二范数距离，也可以替换成 KL 散度，即

   $$\theta_{\text{new}} = \underset{\theta}{\text{argmax}}\, \tilde{L}(\theta|\theta_{\text{now}}); \quad \text{s.t. } \frac{1}{t}\sum_{i=1}^{t}\text{KL}[\pi(\cdot\,|s_i;\theta_{\text{now}})\|\ \pi(\cdot\,|s_i;\theta)] \le \Delta.$$

由于 KL 散度能够衡量两个概率质量函数的接近程度，因此效果一般比球置信域的效果要好。

TRPO 有两个要调的超参数：一个是置信域的半径 Δ，另一个是求解最大化问题中数值算法的学习率。一般而言，Δ 在算法运行过程中要逐渐缩小。

6.2 熵正则

策略学习的目标是学习一个策略网络 $\pi(a|s;\theta)$ 用于控制智能体，我们希望策略网络的输出的概率不要集中在一个动作上，至少要给其他动作一些非零概率。我们可以用熵来衡量概率分布的不确定性，熵小代表概率质量集中，熵大说明随机性很大。

因此我们不妨把熵作为正则项，放在策略学习的目标函数中。策略网络的输出是维度等于 $|\mathcal{A}|$ 的向量，这是动作空间上的离散概率分布，这个概率分布的熵定义为：

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

因此加入正则项后最大化问题变为：

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

带熵正则的最大化问题可以用各种方法求解，其中使用策略梯度方法求关于 θ 的梯度是：

$$g(\theta) \triangleq \nabla_\theta [J(\theta) + \lambda \cdot \mathbb{E}_S[H(S;\theta)]].$$

观测到状态 s，按照策略网络做随机抽样，得到动作 $a \sim \pi(\cdot\,|s;\theta)$。那么

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

是梯度 $g(\theta)$ 的无偏估计。

七、 连续控制

- **离散控制**：动作空间是一个离散的集合。
- **连续控制**：动作空间是一个连续的集合。
  - 方法一：把连续连续空间离散化，那么离散控制的方法就能直接解决连续控制问题。
  - 方法二：直接使用连续控制方法。

7.1 离散控制与连续控制的区别

对动作空间离散化之后, 就可以应用之前学过的方法训练 DQN 或策略网络。但是这样有个缺点, 就是自由度 d 越大，动作空间可执行动作数目越大，且随着 d 指数增长，造成了 **维度灾难**。

7.2 确定策略梯度（DPG）

确定策略梯度（Deterministic Policy Gradient, DPG）是最常用的连续控制方法，也是一种 Actor-Critic 方法，有策略网络（演员）和价值网络（评委）。

7.2.1 策略网络与价值网络

本节的策略网络与前面章节的策略网络不同。

- **之前章节的策略网络 $\pi(a|s;\theta)$ 带有随机性**: 输出结果是包含了 **每一个动作** 的概率值的概率分布向量。
- **本章节的确定策略网络 μ 没有随机性**：对于确定状态 s, 策略网络 μ 输出的动作 a 是确定的 d 维向量。

本节的价值网络 $q(s,a;w)$ 是对动作价值函数 $Q_\pi(s,a)$ 的近似。

7.2.2 算法推导

用行为策略收集经验:

本节的确定策略网络属于异策略（Off-policy）方法，即行为策略可以不同于目标策略。目标策略即确定策略网络 $\mu(s;\theta_{\text{now}})$，其中 θ_{now} 是策略网络最新参数；而行为策略可以是任意的，例如

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

意思是可以使用过时的参数，而且可以加入噪声。

因此我们可以收集轨迹，得到由四元组 s_j, a_j, r_j, s_{j+1} 组成的经验回放数组。

训练策略网络:

根据之前的经验我们可以知道，目标函数是 q 对 μ 打分的相对于状态 s 的期望:

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

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：

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

它是 $\nabla_\theta J(\theta)$ 的无偏估计。g_j 叫做确定策略梯度（Deterministic Policy Gradient），缩写 DPG。

应用链式法则，我们得到下面的定理。

确定策略梯度

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

因此就得到了更新 θ 的算法，每次从经验回放数组里随机抽取一个状态，记作 s_j。计算 $\hat{a}_j = \mu(s_j;\theta)$。用梯度上升更新一次 θ：

$$\theta \leftarrow \theta + \beta \cdot \nabla_\theta \mu(s_j;\theta) \cdot \nabla_a q(s_j, \hat{a}_j; w)$$

训练价值网络：

基本上和前面的一致。

7.2.3 训练流程

每次从经验回放数组中抽取一个四元组，记作 (s_j, a_j, r_j, s_{j+1})。把神经网络当前参数记作 w_{now} 和 θ_{now}。执行下列步骤进行更新：

1. 让策略网络做预测：

$$\hat{a}_j = \mu(s_j;\theta) \text{ 和 } \hat{a}_{j+1} = \mu(s_{j+1};\theta).$$

   注：计算动作 $\hat{a}_j$ 用的时当前的策略网络 $\mu(s_j;\theta_{\text{now}})$，用 $\hat{a}_j$ 来更新 θ_{now}；而从经验回放数组中抽取的 a_j 则是用过时的策略网络 $\mu(s_j;\theta_{\text{old}})$ 计算出来的，用 a_j 来更新 w_{now}，需要注意两者的区别。
2. 让价值网络做预测：

$$\hat{q}_j = q(s_j, a_j; w_{\text{now}}) \text{ 和 } \delta_j = \hat{q}_j - \hat{y}_j.$$

3. 计算 TD 目标和 TD 误差：

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

4. 更新价值网络：

$$w_{\text{new}} \leftarrow w_{\text{now}} - \alpha \cdot \delta_j \cdot \nabla_w q(s_j, a_j; w_{\text{now}}).$$

5. 更新策略网络：

$$\theta_{\text{new}} \leftarrow \theta_{\text{now}} + \beta \cdot \nabla_\theta \mu\big(s_j;\theta_{\text{now}}\big) \cdot \nabla_a q\big(s_j,\hat{a}_j;w_{\text{now}}\big).$$

疑问

但是这样做感觉也很奇怪，SARSA 也不是不能改成这样来用经验回放数组，反正 $\hat{a}_{j+1}$ 每次都是要重新算的，那这里的经验回放不就只剩下一个随机打乱当前状态的作用了嘛，感觉效果确实不会变好。

7.3 深入分析 DPG

1. 即使用了经验回放数组，这里的 $q(s,a;w)$ 近似的仍然是动作价值函数 $Q_\pi(s,a)$ 而不是最优动作价值函数 $Q_\star(s,a)$。但是训练 DPG 的目的也是让 $\mu(s;\theta)$ 趋向于最优策略 $\pi^\star$，在理想情况的话。
2. 价值网络 $q(s,a;w)$ 近似的是 $Q_\pi(s,a)$，其中的 π 是目标策略而不是行为策略。而行为策略影响到的只是经验回放数组的收集效果，不会对训练的最终目标产生影响。极端一点说，你使用随机策略当成行为策略收集数据，也只是会影响拟合的速度罢了。
3. 本质上 DQN 最后的决策方式是 $a_t = \operatorname{argmax}_{a\in\mathcal{A}} Q(s_t,a;w)$，对于离散动作空间很容易实现，但是对连续动作空间就不好做了，所以这时候我们就可以把 DPG 看作对最优动作价值函数 $Q_\star(s,a)$ 的另一种近似方式。我们可以把 μ 和 q 看作 $Q_\star$ 的近似分解。

$$q(s,\mu(s;\theta);w) \approx \max_{a\in\mathcal{A}} Q_\star(s,a)$$

4. DPG 也是一个将目标策略 μ 进行最大化拟合价值函数，进而会有最大化造成高估和自举造成偏差传播，也即高估问题。

疑问

也就是说高估问题本质上来源于最大化 max 操作，而最大化 max 操作又来源于 **确定性**，所以可以说：**高估问题是由确定性策略函数带来的?**

7.4 双延时确定策略梯度（TD3）

我们不用改变神经网络结构，只需要从 DPG 换成 Twin Delayed Deep Deterministic Policy Gradient（TD3）即可大幅提升算法表现。

7.4.1 解决高估问题

第一种方案是目标网络，可以添加两个目标网络 $q(s,a;w^-)$ 和 $\mu(s;\theta^-)$.

但是依然有严重的高估问题。

第二种方案是截断双 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：

$$q(s,a;w_1), \quad q(s,a;w_2), \quad \mu(s;\theta)$$

三个神经网络各对应一个目标网络：

$$q(s,a;w_1^-),\quad q(s,a;w_2^-),\quad \mu(s;\theta^-)$$

用目标策略网络计算动作：

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

然后用两个目标价值网络计算：

$$\widehat{y}_{j,1} = r_j + \gamma \cdot q\big(s_{j+1},\widehat{a}_{j+1}^-;w_1^-\big)$$

$$\widehat{y}_{j,2} = r_j + \gamma \cdot q\big(s_{j+1},\widehat{a}_{j+1}^-;w_2^-\big)$$

取两者较小者为 TD 目标：

$$\widehat{y}_j = \min\Big\{\widehat{y}_{j,1},\widehat{y}_{j,2}\Big\}$$

疑问

本质上就是避免极端情况过大的 TD 目标 $\widehat{y}_j$ 由于自举被传播出去？借助两个参数初始值不同的随机性增加稳健性。

7.4.2 往动作中加入噪声

将用目标网络计算动作：

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

改成：

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

其中噪声 ξ 是从截断正态分布（Clipped Normal Distribution）中抽取的，这是为了避免噪声过大。

疑问

给选取动作时添加一定的随机性以增加稳健性?

7.4.3 减少更新策略网络和目标网络的频率

如果价值网络 q 本身不可靠，那么用价值网络 q 给动作打分是不准确的，无助于改进策略网络 μ。更好的方法是每一轮更新一次价值网络，但是每隔 k 轮更新一次策略网络和三个目标网络，其中 k 是超参数。

7.4.4 训练流程

如果以上的三个技巧都使用了，则就是双延时确定策略梯度，缩写是 TD3。

1. 让目标策略网络做预测：$\widehat{a}_{j+1}^{-}=\mu\left(s_{j+1};\theta_{\text{now}}^{-}\right)+\xi$。其中向量 ξ 的每隔元素都独立从截断正态分布 $\mathcal{CN}(0,\sigma^2,-c,c)$ 中抽取。
2. 让两个目标价值网络做预测：

$$\widehat{q}_{1,j+1}^{-}=q\left(s_{j+1},\widehat{a}_{j+1}^{-};w_{1,\text{now}}^{-}\right)\text{ 和 }\widehat{q}_{2,j+1}^{-}=q\left(s_{j+1},\widehat{a}_{j+1}^{-};w_{2,\text{ now}}^{-}\right)$$

3. 计算 TD 目标：

$$\widehat{y}_j=r_j+\gamma\cdot\min\left\{\widehat{q}_{1,j+1}^{-},\widehat{q}_{1,j+1}^{-}\right\}$$

4. 让两个价值网络做预测：

$$\widehat{q}_{1+j}=q\left(s_j,a_j;w_{1,\text{ now}}\right)\text{ 和 }\widehat{q}_{2,j}=q\left(s_j,a_j;w_{2,\text{ now}}\right)$$

5. 计算 TD 误差：

$$\delta_{1,j}=\widehat{q}_{1,j}-\widehat{y}_j\text{ 和 }\delta_{2,j}=\widehat{q}_{2,j}-\widehat{y}_j$$

6. 更新价值网络：

$$w_{1,\text{ new}}\leftarrow w_{1,\text{ now}}-\alpha\cdot\delta_{1,j}\cdot\nabla_w q\left(s_j,a_j;w_{1,\text{ now}}\right)$$
$$w_{2,\text{ new}}\leftarrow w_{2,\text{ now}}-\alpha\cdot\delta_{2,j}\cdot\nabla_w q\left(s_j,a_j;w_{2,\text{ now}}\right)$$

7. 每隔 k 轮更新一次策略网络和三个目标网络：
   a. 让策略网络做预测：$\widehat{a}_j=\mu\left(s_j;\theta\right)$。然后更新策略网络：

$$\theta_{\text{new}}\leftarrow\theta_{\text{now}}+\beta\cdot\nabla_\theta\mu\left(s_j;\theta_{\text{now}}\right)\cdot\nabla_a q*s_j,\widehat{a}_j;w_{1,\text{ now}}$$

   b. 更新目标网络的参数：

$$\begin{aligned}\theta_{\text{new}}^{-}&\leftarrow\tau\theta_{\text{new}}+(1-\tau)\theta_{\text{now}}^{-},\\ w_{1,\text{ new}}^{-}&\leftarrow\tau w_{1,\text{ new}}+(1-\tau)w_{1,\text{ now}}^{-},\\ w_{2,\text{ new}}^{-}&\leftarrow\tau w_{2,\text{ new}}+(1-\tau)w_{2,\text{ now}}^{-}.\end{aligned}$$

7.5 随机高斯策略

本节用不同于确定策略网络的方法做连续控制。

7.5.1 基本思路

确定策略网络的问题的输出是确定性的，即使神经网络对预测结果的置信度并不高，但也只能被迫选出一个“最佳”的动作。因此一个很自然的想法就是给输出结果增加随机性。但是又不可能多增加太多维度，所以很自然地，我们可以引入一个「方差」的输出维度，进而使用正态分布的概率密度函数来作为策略函数：

$$\pi(a|s) = \frac{1}{\sqrt{2\pi} \cdot \sigma(s)} \cdot \exp\left(-\frac{[a - \mu(s)]^2}{2 \cdot \sigma^2(s)} \right).$$

因此我们需要训练两个神经网络：

$$\mu(s;\theta) \text{ 和 } \rho(s;\theta)$$

后面就是具体的推导细节了。

其中又会碰到不知道动作价值 $Q_\pi(s,a)$ 的问题，不过依然可以通过 REIFORCE 和 Actor-Critic 来解决。

八、 对状态的不完全观测

很多实际应用中，完全观测假设往往不符合实际，例如玩一局游戏，屏幕上的画面并不能反映出游戏的完整状态。

虽然我们可以用当前的观测 o_t 代替状态 s_t，但是效果一定不会好。

一种更合理的方法是让智能体记住过去的观测，这样就能对状态的观测越来越完整，做出越来越理性的决策，也即需要记忆能力，那么策略网络记作：

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

这里就带来了 $o_{1:t}$ 入参大小变化的问题，这时候就不能简单地使用卷积和全连接层，而是要用到循环神经网络了。

九、 模仿学习

- **模仿学习** 不是强化学习，而是强化学习的一种替代品。
- 模仿学习与强化学习有着相同的目的：两者目的都是学习策略网络，从而控制智能体。
- 模仿学习与强化学习有着不同的原理：
  - 模仿学习向人类专家学习，目的是让策略网络做出的决策与人类专家相同；
  - 强化学习利用环境反馈的奖励改进策略，目标是让累积奖励（即回报）最大化。
- 三种常见的模仿学习方法：**行为克隆**（Behavior Cloning）、**逆向强化学习**（Inverse Reinforcement Learning）、**生成式判别模仿学习**（GAIL）。
  - 行为克隆不需要让智能体与环境交互，因此学习「成本」很低。
  - 逆向强化学习、生成式判别模仿学习则需要让智能体与环境交互。

9.1 行为克隆

行为克隆的本质是监督学习（分类或者回归），而不是强化学习。行为克隆通过模仿人类专家的动作来学习策略，而强化学习则是从奖励中学习策略。

模仿学习需要事先准备好的数据集，由（状态，动作）这样的二元组构成，记作：

$$\mathcal{X} = \{(s_1, a_1), \cdots, (s_n, a_n)\}.$$

其中 s_j 是一个状态，而对应的 a_j 是人类专家基于状态 s_j 做出的动作。可以把 s_j 和 a_j 分别视作监督学习中的输入和标签。

9.1.1 连续控制问题

行为克隆用回归的方法训练 **确定策略网络**，然后使用随机梯度下降的方法进行监督学习，直到算法收敛。

9.1.2 离散控制问题

行为克隆用分类的方法训练 **非确定策略网络**，将策略网络 $\pi(a|s;\theta)$ 看做要给多类别分类器，用监督学习的方法训练这个分类器。首先对动作 One-Hot 编码，编码后可以视作离散的概率分布，可以使用交叉熵衡量两个分布的区别。交叉熵定义为

$$H(\overline{a}, f) \triangleq -\sum_{i=1}^{|\mathcal{A}|} \overline{a}_i \cdot \ln f_i$$

向量 $\overline{a}$ 与 f 越接近，它们交叉熵越小，可以用交叉熵作为损失函数，并使用随机梯度下降方法进行训练。

9.1.3 行为克隆与强化学习的对比

行为克隆的本质是监督学习（分类或回归），而不是强化学习，因为行为克隆不需要与环境交互。

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，因此缺乏多样性。智能体面对真实环境，可能遇到陌生状态，因此决策可能很糟糕，进而继续进入另一个奇怪的状态，造成「错误累加」，进入恶性循环。

强化学习效果通常优于行为克隆，强化学习在围棋、电子游戏上的表现可以远超顶级人类玩家，而行为克隆却很难超越人类高手。

强化学习的缺点在于要与环境交互，如果在真实物理世界应用强化学习，要考虑初始化和探索带来的成本。

行为克隆的优势在于离散训练，可以避免与真实环境的交互，不会对环境产生影响。尽管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，这样可以减少有害影响。

9.2 逆向强化学习

逆向强化学习（Inverse Reinforcement Learning，IRL）非常有名，但是今天已经不常用了，下一节介绍的 GAIL 更简单，效果更好。

IRL 的目的是学到一个策略网络 $\pi(a|s;\theta)$，模仿人类专家的黑箱策略 $\pi^\star(a|s)$。IRL 首先从 $\pi^\star(a|s)$ 中学习其隐含的奖励函数 $R^\star$，然后利用奖励函数做强化学习。我们用神经网络 $R(s,a;\rho)$ 来近似奖励函数 $R^\star$。最优策略对应的奖励函数是不唯一的。

9.3 生成判别模仿学习

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。GAIL 还需要收集人类专家的决策记录（即很多条轨迹）。GAIL 的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。

9.3.1 生成判别网络（GAN）

GAIL 的设计基于生成判别网络。生成器和判别器是两个神经网络，生成器负责生成假的样本，判别器负责判定一个样本是真是假。

9.3.2 GAIL 的生成器和判别器

训练数据：GAIL 的训练数据是被模仿的对象（比如人类专家）操作智能体得到的轨迹，记作

$$\tau = [s_1, a_1, s_2, a_2, \cdots, s_m, a_m].$$

数据集中有 k 条轨迹，把数据集记作：

$$\mathcal{X} = \left\{\tau^{(1)}, \tau^{(2)}, \cdots, \tau^{(k)}\right\}.$$

生成器：GAIL 的生成器是策略网络 $\pi(a|s;\theta)$。输入是状态 s，输出是一个向量：

$$f = \pi(\cdot\,|s;\theta)$$

输出向量 f 的维度是动作空间大小 $\mathcal{A}$，它的每个元素对应一个动作，表示执行该动作的概率。给定初始状态 s_1，并让智能体与环境交互，可以得到一条轨迹：

$$\tau = [s_1, a_1, s_2, a_2, \cdots, s_n, a_n].$$

其中动作是根据策略网络抽样得到的：$a_t \sim \pi(\cdot\,|s_t; \theta), \forall t = 1, \cdots, n$；下一时刻的状态是环境根据状态转移函数计算出来的：$s_{t+1} \sim p(\cdot\,|s_t, a_t), \forall t = 1, \cdots, n$。

判别器：GAIL 的判别器记作 $D(s, a; \phi)$，它的输入是状态 s，输出是一个向量：

$$\hat{p} = D(s, \cdot\,|\,\phi).$$

输出向量 $\hat{p}$ 的维度是动作空间的大小 $\mathcal{A}$，它的每个元素对应一个动作 a，把一个元素记作：

$$\hat{p}_a = D(s, a; \phi) \in (0, 1), \quad \forall a \in \mathcal{A}.$$

$\hat{p}_a$ 接近 1 表示 (s, a) 为「真」，即动作 a 是人类专家做的。$\hat{p}_a$ 接近 0 表示 (s, a) 为「假」，即策略网络生成的。

十、 多智能体系统

多智能体系统与单智能体系统的区别：多智能体系统中包含 m 个智能体，智能体之间会相互影响。

多智能体强化学习 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进行交互，利用环境反馈的奖励改进自己的策略，以获得更高的回报（即累积奖励）。

多智能体系统的四种常见设定：合作关系、竞争关系、合作竞争的混合、利己主义。

1. Agent 和环境
   - 什么是 Agent
     - 核心概念: 自治性, 有独立行动的能力
     - Agent 定义: Agent 是处在某个环境中的计算机系统, 该系统有能力在这个环境中自主行动以实现其设计目标.
     - Agent 在与环境的交互过程中, 决定:
       - 选择什么 (what) 动作来执行
       - 什么时候 (when) 执行一个动作
     - 可调整的自治性: 当满足某些条件时, 把决策的控制权交给人
   - 简单的 (无趣的) Agent:
     - 控制系统, 温度控制器, 目标: 保持室内温度, 动作: 加热, 停止加热
     - 软件指示器, UNIX biff 程序, 目标: 监控发给用户的邮件, 动作: 改变屏幕上的图标, 执行一个程序
   - 对象
     - 一个计算实体, 封装了一些状态
     - 可以在这些状态下执行某些动作或者方法
     - 可以通过消息传递进行通信
   - Agent 和对象的区别
     - Agent 是自治的:
       - Agent 包含有更强的自治性的概念
       - Agent 自己决定是否接受其他 Agent 的请求执行一个动作
       - 自治性不是基本的面向对象模型的组成部分
     - Agent 是灵活的:
       - 具有反应的 (reactive), 预动的 (pro-active), 社会的 (social) 行为能力
       - 标准的对象模型根本没有这些行为能力
     - Agent 是主动的:
       - Agent 不是消极的服务提供者
       - 多 Agent 系统本质上是多线程的, 每个 Agent 至少有一个主动控制的线程
   - 专家系统
     - 一类具有专门知识和经验的计算机智能程序系统
     - 模拟人类专家解决领域问题的能力
     - 例子: MYCIN 系统, 可以协助医生处理人类的血液感染问题

- 专家系统 = 知识库 + 推理机
- Agent 和专家系统的区别
  - “经典” 的专家系统是与现实分离的:
    - 它们与任何环境没有直接联系, 而是把用户作为"中间人"发生作用
    - 例子: MYCIN 不直接对病人进行操作
  - 专家系统一般不能采取反应式, 预动的行动
  - 专家系统一般没有社会行为能力, 不能进行合作, 协调和协商
  - 有些专家系统看起来很像是 Agent (特别是执行实时控制任务的专家系统)
- 环境的分类
  - 完全可观察的与部分可观察的
    - 完全可观察的环境: Agent 的传感器在每个时间点上都能获取环境的完整状态
    - 部分可观察的环境: 传感器有噪声, 不精确, 或丢失了部分状态数据
    - 完全不可观察的环境: 没有传感器
    - 部分可观察环境的例子
      - 只有一个本地灰尘传感器的吸尘 Agent 无法知道另一个方格是否有灰尘
      - 自动驾驶出租车无法了解到别的司机在想什么
  - 单 Agent 与多 Agent
    - 单 Agent 环境: 字谜游戏
    - 合作性的多 Agent 环境: 蚁群系统
    - 竞争性的多 Agent 环境: 国际象棋
    - 既合作又竞争的多 Agent 环境: 机器人足球赛
    - 多 Agent 环境中的 Agent 设计问题往往也单 Agent 环境的相差甚远
      - 通信经常作为理性行为出现在多 Agent 环境中
      - 在一些竞争的环境中, 随机行为是理性的
  - 确定的与随机的
    - 确定的环境: 下一个状态完全取决于当前状态和 Agent 执行的动作
    - 否则, 环境是随机的
    - 如果环境是部分可观察的, 那么它可能表现为随机的
    - “随机” 暗示后果是不确定的并且可以用概率来量化
  - 片段式的与序贯式的
    - 在片段式环境中, Agent 的经历被分成了一个个原子片段
      - 在每个片段中 Agent 感知信息并完成单个行动
      - 下一个片段不依赖于以前的片段中采取的行动
      - 例子: 装配线上检验次品零件
    - 在序贯式环境中, 当前的决策会影响到所有未来的决策
      - 行动会有长期的效果
      - 例子: 下棋, 出租车驾驶
    - 片段式的环境更简单, 因为 Agent 不需要前瞻
  - 静态的与动态的
    - 动态的环境: 环境在 Agent 计算的时候会变化
      - 出租车自动驾驶

    - 否则, 环境是静态的
      - 字谜游戏
    - 半动态的环境: 环境本身不随时间变化而变化, 但是 Agent 的性能评价随时间变化
      - 国际象棋 (比赛的时候要计时)
  - 连续的与离散的
    - 环境的状态, 时间的处理方式以及 Agent 的感知信息和行动, 都有离散/连续之分
    - 国际象棋环境
      - 状态是有限的
      - 感知信息和行为是离散的
    - 出租车驾驶环境
      - 连续状态和连续时间
      - 驾驶行为也是连续的
2. 智能 Agent 的属性
  - 智能 Agent 应具有能力:
    - 反应性 (Reactivity)
    - 预动性 (Proactiveness)
    - 社会能力 (Social Ability)
  - 反应性 (Reactivity)
    - 如果一个程序的环境是固定的, 则程序仅仅盲目地执行就可以了
      - 固定环境的例子: 编译程序
    - 真实世界不同: 大多数环境是动态的 (dynamic) 和信息不完全的 (incomplete)
    - 构造适用于动态环境的软件是困难的: 程序需要考虑失败的可能性
    - 一个反应式的 (reactive) 系统能与环境持续的交互, 对环境发生的变化以及时的方式做出反应
  - 预动性 (Proactiveness)
    - 对环境做出反应是容易的
      - 例子: 刺激 → 反应 (规则)
    - 但是, 我们想 Agent 为我们做事情
      - 因此, 表现出目的引导的行为
    - 预动性 = 通过主动发起动作, 表现出目标引导的行为
      - 识别机会
  - 社会能力 (Social Ability)
    - 真实世界是一个多 Agent 环境: Agent 在实现自己目标的同时, 不能不考虑其他 Agent 的目标
      - 一些目标只能通过与其他 Agent 交互来达到
      - 多台计算机构成的环境类似: 英特网是一个很好的例子
    - Agent 的社会能力: 通过合作 (cooperation), 协调 (coordination), 协商 (negotiation).其他 Agent (可能是人).互的能力
      - 至少, 包括通信能力
    - 社会能力: 合作 (Cooperation)
      - 作为一个团队一起工作以达到一个共享的目标

    - 合作经常是由下列事实导致的:
      - 单一的 Agent 不能达到该目标
      - 合作将获得更好的结果 (比如, 更快达到目标)
  - 社会能力: 协调 (Coordination)
    - 如何管理 Agent 活动之间的内部依赖关系
    - 例子: 如果你想使用的资源和我想使用的资源是不能共享的, 则我们需要协调
  - 社会能力: 协商 (Negotiation)
    - Agent 就共同关心的问题达成一致的能力
    - 例子
      - 屋里有一台电视机, 你想看电影, 你丈夫想看足球
      - 一种可能的约定: 今晚看足球, 明晚看电影
  - 通常涉及出价 (offer), 还价 (counter-offer), 参与者在协商过程中会作出让步
- 智能 Agent 的其他属性
  - 移动性 (Mobility): 移动能力
    - 例: 软件 Agent 可以在信息网络环境中运动
  - 诚实性 (Veracity): 不会故意发送错误信息
  - 仁慈性 (Benevolence): Agent 之间的目标不会产生冲突, 每个 Agent 总是尽量完成所要求的任务
  - 合理性 (Rationality): 为了实现目标而努力, 不会采取阻碍目标实现的动作, 至少在它的信念中是这样
  - 学习性 (Learning/Adaption): 随时间改进性能

十一、 注意力机制与多智能体强化学习

注意力机制（Attention）是一种重要的深度学习方法，它最主要的用途是自然语言处理。

为什么要因为注意力机制在 Attention 诞生之前，已经有 CNN 和 RNN 及其变体模型了，那为什么还要引入 Attention 机制？主要有两个方面的原因，如下：

1. 计算能力的限制：当要记住很多“信息“，模型就要变得更复杂，然而目前计算能力依然是限制神经网络发展的瓶颈。
2. 优化算法的限制：LSTM只能在一定程度上缓解RNN中的长距离依赖问题，且信息“记忆”能力并不高。

11.1 自注意力在中心化训练中的应用

自注意力机制（Self-Attention）是改进多智能体强化学习的一种有效技巧。

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。