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

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

3. 对函数值 $f(x_1), ..., f(x_n)$ 求平均，再乘以 $\Omega$ 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 $\Omega$ 满足不等式约束，则其中的重点在于：

1. 在 $\Omega$ 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 $\Omega$ 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 $\Omega$ 上做非均匀随机抽样，得到 $n$ 个样本 $x_1, ..., x_n \sim p(\cdot)$；
2. 对函数值 $f(x_1), ..., f(x_n)$ 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

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

       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）。

1. 从 $q(x)$ 随机抽取一个样本 $x_i$；
2. 从均匀分布 $U(0,1)$ 中随机抽取一个样本 $u_i$；
3. 比较 $u_i$ 与 $\alpha = \dfrac{p(x)}{k \cdot q(x)}$，如果 $u_i \le \alpha$ 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 $\pi$ 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：

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

#### 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 $\pi$ 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 $Q_\pi$，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 $\delta_t$。

### 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 $\varepsilon$-greedy 策略

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

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 $(s_j, a_j, r_j, s_{j+1})$。
   1. 对 DQN 做正向传播，得到 Q 值：
   $$\hat{q}_j = Q(s_j, a_j; w_{\text{now}}) \text{ 和 } \hat{q}_{j+1} = \max_{a\in\mathcal{A}} Q(s_{j+1}, a; w_{\text{now}})$$
   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 做反向传播，得到梯度：
   $$g_j = \nabla_w Q(s_j, a_j; w_{\text{now}})$$
   4. 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 $Q_\pi$ 并不是用于控制智能体，而是用来评价策略函数 $\pi$ 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 $\pi$ 控制智能体，被看作「演员」；而 $Q_\pi$ 评价 $\pi$ 的表现，用于帮忙改进 $\pi$，被看作「评委」。Actor-Critic 通常用 SARSA 训练「评委」$Q_\pi$。

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 $Q_\pi$ 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 $\pi$，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 $\pi$ 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 $w_{\text{now}}$，当前策略为 $\pi_{\text{now}}$。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 $w_{\text{now}}$，当前策略为 $\pi_{\text{now}}$。执行以下步骤更新价值网络和策略。

1. 用策略网络 $\pi_{\text{now}}$ 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 $\pi$。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**
  - $p_j \propto |\delta_j| + \varepsilon$
  - $$p_j \propto \frac{1}{\operatorname{rank}(j)}$$
- 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 $\beta$ 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 $\mathbb{E}_\varepsilon\left[\max_{a\in\mathcal{A}} Q(s,a;w)\right] \geq \max_{a\in\mathcal{A}} Q_\star(s,a)$

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 $\mu + \sigma \circ \xi$，$\mu$、$\sigma$ 分别是均值和方差，而 $\xi$ 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 $\mu$ 也没关系，只要在参数 $\mu$ 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

TODO