第六章 价值学习高级技巧

第 4 章介绍了 DQN，并且用 Q 学习算法（一种 TD 算法）训练 DQN。如果读者按照第 4 章最原始的方式实现 DQN，效果会很不理想。想要提升 DQN 的表现，需要用本章的高级技巧。文献中已经有充分实验结果表明这些高级技巧对 DQN 非常有效，而且这些技巧不冲突，可以一起使用。这些技巧并不局限于 DQN，而是可以用于多种价值学习和策略学习方法。

第 6.1、6.2 节介绍两种方法改进 TD 算法，让 DQN 训练得更好。第 6.1 节介绍经验回放 (Experience Replay) 和优先经验回放 (Prioritized Experience Replay)。第 6.2 节讨论 DQN 的高估问题以及解决方案——目标网络 (Target Network) 和双 Q 学习算法 (Double Q-learning)。

第 6.3、6.4 节介绍两种方法改进 DQN 神经网络结构（不是对 TD 算法的改进）。第 6.3 节介绍对决网络 (Dueling Network)，它把动作价值 (Action-Value) 分解成状态价值 (State-Value) 与优势 (Advantage)。第 6.4 节介绍噪声网络 (Noisy Net)，它往神经网络的参数中加入随机性，鼓励探索。

6.1 经验回放

经验回放 (Experience Replay) 是强化学习中一个重要的技巧，可以大幅提升强化学习的表现。经验回放的意思是把智能体与环境交互的记录（即经验）储存到一个数组里，事后反复利用这些经验训练智能体。这个数组被称为经验回放数组 (Replay Buffer)。

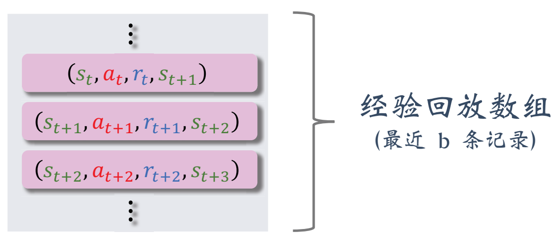


图 6.1: 经验回放数组。

具体来说，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，存入一个数组。需要人为指定数组的大小（记作 b）。数组中只保留最近 b 条数据；当数组存满之后，删除掉最旧的数据。数组的大小 b 是个需要调的超参数，会影响训练的结果；通常设置 b 为 $10^5 \sim 10^6$。

6.1.1 经验回放的优点

经验回放的一个好处在于打破序列的相关性。训练 DQN 的时候，每次我们用一个四元组对 DQN 的参数做一次更新。我们希望相邻两次使用的四元组是独立的。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 $(s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2})$ 有很强的相关性。依次使用这些强关联的四元组训练 DQN，效果往往会很差。经验回放每次从数组里随机抽取一个四元组，用来对 DQN 参数做一次更新。这样随机抽到的四元组都

是独立的，消除了相关新。

经验回放的另一个好处是重复利用收集到的经验，而不是用一次就丢弃，这样可以用更少的样本数量达到同样的表现。重复利用经验、不重复利用经验的收敛曲线通常如图 6.2 所示。图的横轴是样本数量，纵轴是平均回报。

注 在阅读文献的时候请注意“样本数量” (Sample Complexity) 与“更新次数”两者的区别。样本数量是指智能体从环境中获取的奖励 r 的数量。而一次更新的意思是从经验回放数组里取出一个或多个四元组，用它对参数 $\boldsymbol{w}$ 做一次更新。通常来说，样本数量更重要，因为在实际应用中收集经验比较困难；比如，在机器人的应用中，需要在现实世界里做一次实验才能收集到一条经验。做更新的次数不是那么重要，更新次数只会影响训练时的计算量而已。

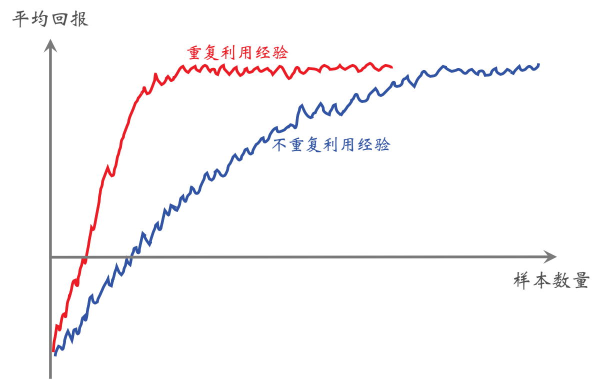


图 6.2: 收敛曲线示意图。

6.1.2 经验回放的局限性

需要注意，并非所有的强化学习方法都允许重复使用过去的经验。经验回放数组里的数据全都是用行为策略 (Behavior Policy) 控制智能体收集到的。在收集经验同时，我们也在不断地改进策略。策略的变化导致收集经验时用的行为策略是过时的策略，不同于当前我们想要更新的策略——即目标策略 (Target Policy)。也就是说，经验回放数组中的经验通常是过时的行为策略收集的，而我们真正想要学的目标策略不同于过时的行为策略。

有些强化学习方法允许行为策略不同于目标策略。这样的强化学习方法叫做**异策略** (Off-policy)。比如 Q 学习、确定策略梯度 (DPG) 都属于异策略。由于它们允许行为策略不同于目标策略，因此过时行为策略收集到的经验可以被重复利用。**经验回放适用于异策略。**

有些强化学习方法要求行为策略与目标策略必须相同。这样的强化学习方法叫做**同策略** (On-policy)。比如 SARSA、REINFORCE、A2C 都属于同策略。它们要求经验必须是当前的目标策略收集到的，而不能使用过时的经验。**经验回放不适用于同策略。**

6.1.3 优先经验回放

优先经验回放 (Prioritized Experience Replay) 是一种特殊的经验回放方法，它比普通的经验回放效果更好：既能让收敛更快，也能让收敛时的平均回报更高。经验回放数组里有 b 个四元组，普通经验回放每次均匀抽样得到一个样本——即四元组 (s_j, a_j, r_j, s_{j+1})，用它来更新 DQN 的参数。优先经验回放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 $Q(s_j, a_j; \boldsymbol{w})$ 离 $Q_\star(s_j, a_j)$ 较远，则四元组 (s_j, a_j, r_j, s_{j+1}) 应当有较高的权重。

为什么样本的重要性会有所不同呢？设想你用强化学习训练一辆无人车。经验回放数组中的样本绝大多数都是车辆正常行驶的情形，只有极少数样本是意外情况，比如旁边车辆强行变道、行人横穿马路、警察封路要求绕行。数组中的样本的重要性显然是不同的。车辆正常行驶的样本要多少有多少，而且正常行驶的情形很容易处理，出错的可能性非常小。意外情况的样本非常少，但是又及其重要，处理不好就会车毁人亡。所以意外情况的样本应当有更高的权重，受到更多关注。不应该同等对待正常行驶、意外情况的样本。

如何自动判断哪些样本更重要呢？举个例子，自动驾驶中的意外情况数量少、而且难以处理，导致 DQN 的预测 $Q(s_j, a_j; \boldsymbol{w})$ 严重偏离真实价值 $Q_\star(s_j, a_j)$。因此，要是 $\big|Q(s_j, a_j; \boldsymbol{w}) - Q_\star(s_j, a_j)\big|$ 较大，则应该给样本 (s_j, a_j, r_j, s_{j+1}) 较高的权重。然而实际上我们无从得知 $\big|Q(s_j, a_j; \boldsymbol{w}) - Q_\star(s_j, a_j)\big|$；不妨把它替换成 TD 误差。回忆一下，TD 误差的定义是：

$$\delta_j \ \triangleq\ Q\big(s_j, a_j; \boldsymbol{w}_{\text{now}}\big) \ - \ \underbrace{\Big[r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q\big(s_{j+1}, a; \boldsymbol{w}_{\text{now}}\big)\Big]}_{\text{即 TD 目标}}.$$

如果 TD 误差的绝对值 $|\delta_j|$ 大，说明当前的 DQN（参数是 $\boldsymbol{w}_{\text{now}}$）对 (s_j, a_j) 的真实价值的评估不准确，那么应该给 (s_j, a_j, r_j, s_{j+1}) 设置较高的权重。

优先经验回放对数组里的样本做非均匀抽样。四元组 (s_j, a_j, r_j, s_{j+1}) 的权重是 TD 误差的绝对值 $|\delta_j|$，它的抽样概率取决于 TD 误差。有两种方法设置抽样概率。一种抽样概率是：

$$p_j \ \propto\ |\delta_j| + \epsilon.$$

此处的 ϵ 是个很小的数，防止抽样概率接近零，用于保证所有样本都以非零的概率被抽到。另一种抽样方式先对 $|\delta_j|$ 做降序排列，然后计算

$$p_j \ \propto\ \frac{1}{\text{rank}(j)}.$$

此处的 $\text{rank}(j)$ 是 $|\delta_j|$ 的序号；大的 $|\delta_j|$ 的序号小，小的 $|\delta_j|$ 的序号大。两种方式的原理是一样的：TD 误差大的样本被抽样到的概率大。

优先经验回放做非均匀抽样，四元组 (s_j, a_j, r_j, s_{j+1}) 被抽到的概率是 p_j。抽样是非均匀的，不同的样本有不同的抽样概率，这样会导致 DQN 的预测有偏差。应该相应调整学习率，抵消掉不同抽样概率造成的偏差。TD 算法用“随机梯度下降”来更新参数：

$$\boldsymbol{w}_{\text{new}} \ \leftarrow\ \boldsymbol{w}_{\text{now}} - \alpha \cdot \boldsymbol{g},$$

此处的 α 是学习率，$\boldsymbol{g}$ 是损失函数关于 $\boldsymbol{w}$ 的梯度。如果用均匀抽样，那么所有样本有相同的学习率 α。如果做非均匀抽样的话，应该根据抽样概率来调整学习率 α；如果一条样本被抽样的概率大，那么它的学习率就应该比较小。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[1]。

注 均匀抽样是一种特例，即所有抽样概率都相等：$p_1 = \cdots = p_b = \frac{1}{b}$。在这种情况下，有 $(b \cdot p_j)^\beta = 1$，因此学习率都相同：$\alpha_1 = \cdots = \alpha_b = \alpha$。

注 读者可能会问下面的问题。如果样本 (s_j, a_j, r_j, s_{j+1}) 很重要，它被抽到的概率 p_j 很大，可是它的学习率却很小。当 $\beta = 1$ 时，如果抽样概率 p_j 变大 10 倍，则学习率 α_j 减小 10 倍。抽样概率、学习率两者岂不是抵消了吗？优先经验回放有什么意义呢？两者其实并没有抵消，因为下面两种方式并不等价：

- 设置学习率为 α，使用样本 (s_j, a_j, r_j, s_{j+1}) 计算一次梯度，更新一次参数 $\boldsymbol{w}$；
- 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 $\boldsymbol{w}$。

乍看起来两种方式区别不大，但其实第二种方式是对样本更有效的利用。第二种方式的缺点在于计算量大了十倍；所以第二种方式只被用于重要的样本。

序号	四元组	TD 误差	抽样概率	学习率
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$j-1$	$(s_{j-1}, a_{j-1}, r_{j-1}, s_j)$	δ_{j-1}	$p_{j-1} \propto \lvert\delta_{j-1}\rvert + \epsilon$	$\alpha \cdot (b \cdot p_{j-1})^{-\beta}$
j	(s_j, a_j, r_j, s_{j+1})	δ_j	$p_j \propto \lvert\delta_j\rvert + \epsilon$	$\alpha \cdot (b \cdot p_j)^{-\beta}$
$j+1$	$(s_{j+1}, a_{j+1}, r_{j+1}, s_{j+2})$	δ_{j+1}	$p_{j+1} \propto \lvert\delta_{j+1}\rvert + \epsilon$	$\alpha \cdot (b \cdot p_{j+1})^{-\beta}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

图 6.3: 优先经验回放数组。

优先经验回放数组如图 6.3 所示。设 b 为数组大小，需要手动调整。如果样本（即四元组）的数量超过了 b，那么要删除最旧的样本。数组里记录了四元组、TD 误差、抽样概率、以及学习率。注意，数组里存的 TD 误差 δ_j 是用过时 DQN 参数计算出来的：

$$\delta_j \;=\; Q(s_j, a_j; \boldsymbol{w}_{\text{old}}) \;-\; \Big[r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q(s_{j+1}, a; \boldsymbol{w}_{\text{old}}) \Big].$$

做经验回放的时候，每次取出一个（或多个）四元组，用它计算出新的 TD 误差：

$$\delta'_j \;=\; Q(s_j, a_j; \boldsymbol{w}_{\text{now}}) \;-\; \Big[r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q(s_{j+1}, a; \boldsymbol{w}_{\text{now}}) \Big]$$

然后用它更新 DQN 的参数。用这个新的 δ'_j 取代数组中旧的 δ_j。

[1]论文里建议一开始让 β 比较小，最终增长到 1。

6.2 高估问题及解决方法

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，而且高估通常是非均匀的。这个缺陷导致 DQN 的表现很差。高估问题并不是 DQN 本身的缺陷，而是训练 DQN 用的 Q 学习算法的缺陷。Q 学习产生高估的原因有两个：第一，自举导致偏差的传播；第二，最大化导致 TD 目标高估真实价值。为了缓解高估，需要从导致高估的两个原因下手，改进 Q 学习算法。双 Q 学习算法是一种有效的改进，可以大幅缓解高估及其危害。

6.2.1 自举导致偏差的传播

在强化学习中，自举意思是“用一个估算去更新同类的估算”，类似于“自己把自己给举起来”。我们在第 5.4 节讨论过 SARSA 算法中的自举。下面回顾训练 DQN 用的 Q 学习算法，研究其中存在的自举。算法每次从经验回放数组 (Replay Buffer) 中抽取一个四元组 (s_j, a_j, r_j, s_{j+1})。然后执行以下步骤，对 DQN 的参数做一轮更新：

1. 计算 TD 目标：
$$\widehat{y}_j \;=\; r_j + \gamma \cdot \underbrace{\max_{a_{j+1}\in\mathcal{A}} Q\big(s_{j+1}, a_{j+1};\, \boldsymbol{w}_{\text{now}}\big)}_{\text{DQN 自己做出的估计}}.$$

2. 定义损失函数
$$L(\boldsymbol{w}) \;=\; \frac{1}{2}\Big[\underbrace{Q(s_j, a_j; \boldsymbol{w}) \;-\; \widehat{y}_j}_{\text{让 DQN 拟合 } \widehat{y}_j}\Big]^2.$$

3. 把 $\widehat{y}_j$ 看做常数，做一次梯度下降更新参数：
$$\boldsymbol{w}_{\text{new}} \;\leftarrow\; \boldsymbol{w}_{\text{now}} \;-\; \alpha \cdot \nabla_{\boldsymbol{w}} L(\boldsymbol{w}_{\text{now}}).$$

第一步中的 TD 目标 $\widehat{y}_j$ 部分基于 DQN 自己做出的估计；第二步让 DQN 去拟合 $\widehat{y}_j$。这就意味着我们用了 DQN 自己做出的估计去更新 DQN 自己，这属于自举。

自举对 DQN 的训练有什么影响呢？$Q(s, a; \boldsymbol{w})$ 是对价值 $Q_\star(s, a)$ 的近似；最理想的情况下，$Q(s, a; \boldsymbol{w}) = Q_\star(s, a)$，$\forall\, s, a$。假如碰巧 $Q(s_{j+1}, a_{j+1}; \boldsymbol{w})$ 低估（或高估）真实价值 $Q_\star(s_{j+1}, a_{j+1})$，则会发生下面的情况：

$$\begin{array}{rccc}
 & Q\big(s_{j+1}, a_{j+1}; \boldsymbol{w}\big) & \text{低估（或高估）} & Q_\star\big(s_{j+1}, a_{j+1}\big) \\
\Longrightarrow & \widehat{y}_j & \text{低估（或高估）} & Q_\star\big(s_j, a_j\big) \\
\Longrightarrow & Q\big(s_j, a_j; \boldsymbol{w}\big) & \text{低估（或高估）} & Q_\star\big(s_j, a_j\big).
\end{array}$$

结论 6.1. 自举导致偏差的传播

如果 $Q(s_{j+1}, a_{j+1}; \boldsymbol{w})$ 是对真实价值 $Q_\star(s_{j+1}, a_{j+1})$ 的低估（或高估），就会导致 $Q(s_j, a_j; \boldsymbol{w})$ 低估（或高估）价值 $Q_\star(s_j, a_j)$。也就是说低估（或高估）从 (s_{j+1}, a_{j+1}) 传播到 (s_j, a_j)，让更多的价值被低估（或高估）。♣

6.2.2 最大化导致高估

首先用数学解释为什么最大化会导致高估。设 $x_1, \cdots, x_d$ 为任意 d 个实数。往 $x_1, \cdots, x_d$ 中加入任意均值为零的随机噪声，得到 $Z_1, \cdots, Z_d$，它们是随机变量，随机性来源于随机噪声。很容易证明均值为零的随机噪声不会影响均值：

$$\mathbb{E}\Big[\operatorname{mean}\big(Z_1, \cdots, Z_d\big)\Big] \;=\; \operatorname{mean}\big(x_1, \cdots, x_d\big).$$

用稍微复杂一点的证明，可以得到：

$$\mathbb{E}\Big[\max\big(Z_1, \cdots, Z_d\big)\Big] \;\geq\; \max\big(x_1, \cdots, x_d\big).$$

公式中的期望是关于噪声求的。这个不等式意味着先加入均值为零的噪声，然后求最大值，会产生高估。

假设对于所有的动作 $a \in \mathcal{A}$ 和状态 $s \in \mathcal{S}$，DQN 的输出是真实价值 $Q_\star(s,a)$ 加上均值为零的随机噪声 ϵ：

$$Q\big(s, a; \boldsymbol{w}\big) \;=\; Q_\star\big(s, a\big) + \epsilon.$$

显然 $Q(s,a;\boldsymbol{w})$ 是对真实价值 $Q_\star(s,a)$ 的无偏估计。然而有这个不等式：

$$\mathbb{E}_\epsilon\Big[\max_{a\in\mathcal{A}} Q\big(s, a; \boldsymbol{w}\big)\Big] \;\geq\; \max_{a\in\mathcal{A}} Q_\star\big(s, a\big).$$

公式说明尽管 DQN 是对真实价值的无偏估计，但如果求最大化，DQN 则会高估真实价值。复习一下，TD 目标是这样算出来的：

$$\widehat{y}_j \;=\; r_j \;+\; \gamma \cdot \underbrace{\max_{a\in\mathcal{A}} Q\big(s_{j+1}, a; \boldsymbol{w}\big)}_{\text{高估 } \max_{a\in\mathcal{A}} Q_\star(s_{j+1}, a)}.$$

这说明 TD 目标 $\widehat{y}_j$ 通常是对真实价值 $Q_\star(s_j,a_j)$ 的高估。TD 算法鼓励 $Q(s_j,a_j;\boldsymbol{w})$ 接近 TD 目标 $\widehat{y}_j$，这会导致 $Q(s_j,a_j;\boldsymbol{w})$ 高估真实价值 $Q_\star(s_j,a_j)$。

结论 6.2. 最大化导致高估

即使 DQN 是真实价值 $Q_\star$ 的无偏估计，只要 DQN 不恒等于 $Q_\star$，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 就会出现高估。 ♣

6.2.3 高估的危害

我们为什么要避免高估？高估真的有害吗？高估本身是无害的，除非高估是非均匀的。举个例子，动作空间是 $\mathcal{A} = \{左, 右, 上\}$。给定当前状态 s，每个动作有一个真实价值：

$$Q_\star\big(s, 左\big) \;=\; 200, \qquad Q_\star\big(s, 右\big) \;=\; 100, \qquad Q_\star\big(s, 上\big) \;=\; 230.$$

智能体应当选择动作“上”，因为“上”的价值最高。假如高估是均匀的，所有的价值都被高估了 100：

$$Q\big(s, 左; \boldsymbol{w}\big) \;=\; 300, \qquad Q\big(s, 右; \boldsymbol{w}\big) \;=\; 200, \qquad Q\big(s, 上; \boldsymbol{w}\big) \;=\; 330.$$

那么动作“上”仍然有最大的价值，智能体会选择“上”。这个例子说明高估本身不是问题，只要所有动作价值被同等高估。

但实践中，所有的动作价值会被同等高估吗？每当取出一个四元组 (s, a, r, s') 用来更新一次 DQN，就很有可能加重 DQN 对 $Q_\star(s, a)$ 的高估。对于同一个状态 s，三种组合 $(s, \text{左})$、$(s, \text{右})$、$(s, \text{上})$ 出现在经验回放数组中的频率是不同的，所以三种动作被高估的程度是不同的。假如动作价值被高估的程度不同，比如

$$Q(s, \text{左};\, \boldsymbol{w}) \;=\; 280, \qquad Q(s, \text{右};\, \boldsymbol{w}) \;=\; 300, \qquad Q(s, \text{上};\, \boldsymbol{w}) \;=\; 260,$$

那么智能体做出的决策就是向右走，因为“右”的价值貌似最高。但实际上“右”是最差的动作，它的实际价值低于其余两个动作。

综上所述，用 Q 学习算法训练 DQN 总会导致 DQN 高估真实价值。对于多数的 $s \in \mathcal{S}$ 和 $a \in \mathcal{A}$，有这样的不等式：

$$Q(s, a; \boldsymbol{w}) \;>\; Q_\star(s, a).$$

高估本身不是问题，真正的麻烦在于 DQN 的高估往往是非均匀的。如果 DQN 有非均匀的高估，那么用 DQN 做出的决策是不可靠的。我们已经分析过导致高估的原因：

- TD 算法属于“自举”，即用 DQN 的估计值去更新 DQN 自己。自举会导致偏差的传播。如果 $Q(s_{j+1}, a_{j+1}; \boldsymbol{w})$ 是对 $Q_\star(s_{j+1}, a_{j+1})$ 的高估，那么高估会传播到 (s_j, a_j)，让 $Q(s_j, a_j; \boldsymbol{w})$ 高估 $Q_\star(s_j, a_j)$。自举导致 DQN 的高估从一个二元组 (s, a) 传播到更多的二元组。
- TD 目标 $\widehat{y}$ 中包含一项最大化，这会导致 TD 目标高估真实价值 $Q_\star$。Q 学习算法鼓励 DQN 的预测接近 TD 目标，因此 DQN 会高估 $Q_\star$。

找到了产生高估的原因，就可以想办法解决问题。**想要避免 DQN 的高估，要么切断“自举”，要么避免最大化造成高估**。注意，高估并不是 DQN 自身的属性；高估纯粹是算法造成的。想要避免高估，就要用更好的算法替代原始的 Q 学习算法。

6.2.4 使用目标网络

上文已经讨论过，切断“自举”可以避免偏差的传播，从而缓解 DQN 的高估。回顾一下，Q 学习算法这样计算 TD 目标：

$$\widehat{y}_j \;=\; r_t \;+\; \underbrace{\gamma \cdot \max_{a \in \mathcal{A}} Q\big(s_{j+1}, a;\, \boldsymbol{w}\big)}_{\text{DQN 做出的估计}}.$$

然后做梯度下降更新 $\boldsymbol{w}$，使得 $Q(s_j, a_j; \boldsymbol{w})$ 更接近 $\widehat{y}_j$。想要切断自举，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。另一个神经网络就被称作**目标网络** (Target Network)。把目标网络记作：

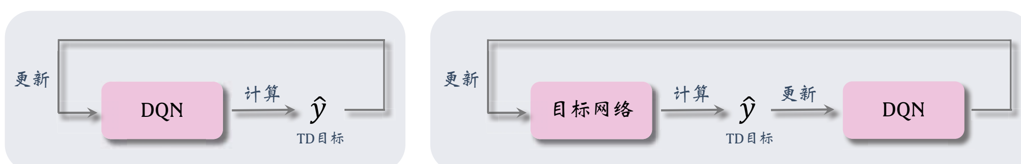
$$Q\big(s, a;\, \boldsymbol{w}^-\big).$$

它的神经网络结构与 DQN 完全相同，但是参数 $\boldsymbol{w}^-$ 不同于 $\boldsymbol{w}$。

使用目标网络的话，Q 学习算法的用下面的方式实现。每次随机从经验回放数组中取一个四元组，记作 (s_j, a_j, r_j, s_{j+1})。设 DQN 和目标网络当前的参数分别为 $\boldsymbol{w}_{\text{now}}$ 和

$\boldsymbol{w}_{\text{now}}^-$，执行下面的步骤对参数做一次更新：

1. 对 DQN 做正向传播，得到：
$$\widehat{q}_j \;=\; Q\big(s_j, a_j;\, \boldsymbol{w}_{\text{now}}\big).$$
2. 对目标网络做正向传播，得到
$$\widehat{q}_{j+1}^- \;=\; \max_{a\in\mathcal{A}} Q\big(s_{j+1}, a;\, \boldsymbol{w}_{\text{now}}^-\big).$$
3. 计算 TD 目标和 TD 误差：
$$\widehat{y}_j^- \;=\; r_j + \gamma\cdot\widehat{q}_{j+1}^- \qquad \text{和} \qquad \delta_j \;=\; \widehat{q}_j - \widehat{y}_j^- .$$
4. 对 DQN 做反向传播，得到梯度 $\nabla_{\boldsymbol{w}}\, Q\big(s_j, a_j;\, \boldsymbol{w}_{\text{now}}\big)$。
5. 做梯度下降更新 DQN 的参数：
$$\boldsymbol{w}_{\text{new}} \;\leftarrow\; \boldsymbol{w}_{\text{now}} \;-\; \alpha\,\cdot\,\delta_j\,\cdot\,\nabla_{\boldsymbol{w}}\, Q\big(s_j, a_j;\, \boldsymbol{w}_{\text{now}}\big).$$
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：
$$\boldsymbol{w}_{\text{new}}^- \;\leftarrow\; \tau\cdot\boldsymbol{w}_{\text{new}} \;+\; \big(1-\tau\big)\cdot\boldsymbol{w}_{\text{now}}^-.$$



原始的 Q 学习算法　　　　使用目标网络的 Q 学习算法

图 6.4

如图 6.4(左) 所示，原始的 Q 学习算法用 DQN 计算 $\widehat{y}$，然后拿 $\widehat{y}$ 更新 DQN 自己，造成自举。如图 6.4(右) 所示，可以改用目标网络计算 $\widehat{y}$，这样就避免了用 DQN 的估计更新 DQN 自己，降低自举造成的危害。然而这种方法并不可能完全避免自举，原因是目标网络的参数仍然与 DQN 相关。

6.2.5 双 Q 学习算法

造成 DQN 高估的原因不是 DQN 模型本身的缺陷，而是训练 DQN 所用的算法有不足之处：第一，自举造成偏差的传播；第二，最大化造成 TD 目标的高估。在 Q 学习算法中使用目标网络，可以缓解自举造成的偏差，但是无助于缓解最大化造成的高估。本小节介绍**双 Q 学习 (Double Q Learning)** 算法，它在目标网络的基础上做改进，缓解最大化造成的高估。

注 本小节介绍的双 Q 学习算法在文献中被称作 Double DQN，缩写 DDQN。本书不采用 DDQN 这名字，因为这个名字比较误导。双 Q 学习（即所谓的 DDQN）只是一种 **TD 算法**而已，它可以把 DQN 训练得更好。双 Q 学习并没有用区别于 DQN 的**模型**。本节中的模型只有一个，就是 DQN。而我们所讨论的只是训练 DQN 的三种 TD 算法：原始的 Q 学习、用目标网络的 Q 学习、以及双 Q 学习。

为了解释原始 Q 学习、用目标网络的 Q 学习、以及双 Q 学习三者的区别，我们再回

顾一下 Q 学习算法中的 TD 目标：

$$\widehat{y}_j \ = \ r_j \ + \ \gamma \cdot \max_{a\in\mathcal{A}} Q\big(s_{j+1}, a;\, \boldsymbol{w}\big).$$

不妨把最大化拆成两步：

1. **选择**——即基于状态 s_{j+1}，选出一个动作使得 DQN 的输出最大化：

$$a^{\star} \ = \ \underset{a\in\mathcal{A}}{\operatorname{argmax}} \ Q\big(s_{j+1}, a;\, \boldsymbol{w}\big).$$

2. **求值**——即计算 $(s_{j+1}, a^{\star})$ 的价值，从而算出 TD 目标：

$$\widehat{y}_j \ = \ r_j + Q\big(s_{j+1}, a^{\star};\, \boldsymbol{w}\big).$$

以上是原始的 Q 学习算法，选择和求值都用 DQN。上一小节改进了 Q 学习，选择和求值都用目标网络：

$$\begin{aligned} \textbf{选择：} \qquad & a^{-} \ = \ \underset{a\in\mathcal{A}}{\operatorname{argmax}} \ Q\big(s_{j+1}, a;\, \boldsymbol{w}^{-}\big), \\ \textbf{求值：} \qquad & \widehat{y_t^{-}} \ = \ r_t + Q\big(s_{j+1}, a^{-};\, \boldsymbol{w}^{-}\big). \end{aligned}$$

本小节介绍**双 Q 学习**，第一步的选择用 DQN，第二步的求值用目标网络：

$$\begin{aligned} \textbf{选择：} \qquad & a^{\star} \ = \ \underset{a\in\mathcal{A}}{\operatorname{argmax}} \ Q\big(s_{j+1}, a;\, \boldsymbol{w}\big), \\ \textbf{求值：} \qquad & \widetilde{y}_t \ = \ r_t + Q\big(s_{j+1}, a^{\star};\, \boldsymbol{w}^{-}\big). \end{aligned}$$

为什么双 Q 学习可以缓解最大化造成的高估呢？不难证明出这个不等式：

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

因此，

$$\underbrace{\widetilde{y}_t}_{\text{双 Q 学习}} \quad \leq \quad \underbrace{\widehat{y_t^{-}}}_{\text{用目标网络的 Q 学习}} .$$

这个公式说明双 Q 学习得到的 TD 目标更小；也就是说，与用目标网络的 Q 学习相比，双 Q 学习缓解了高估。

双 Q 学习算法的流程如下。每次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})。设 DQN 和目标网络当前的参数分别为 $\boldsymbol{w}_{\text{now}}$ 和 $\boldsymbol{w}_{\text{now}}^{-}$，执行下面的步骤对参数做一次更新：

1. 对 DQN 做正向传播，得到：

$$\widehat{q}_j \ = \ Q\big(s_j, a_j;\, \boldsymbol{w}_{\text{now}}\big).$$

2. 选择：

$$a^{\star} \ = \ \underset{a\in\mathcal{A}}{\operatorname{argmax}} \, Q\big(s_{j+1}, a;\, \boldsymbol{w}_{\text{now}}\big).$$

3. 求值：

$$\widehat{q}_{j+1} \ = \ Q\big(s_{j+1}, a^{\star};\, \boldsymbol{w}_{\text{now}}^{-}\big).$$

4. 计算 TD 目标和 TD 误差：

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

5. 对 DQN 做反向传播，得到梯度 $\nabla_{\boldsymbol{w}} Q(s_j, a_j; \boldsymbol{w}_{\text{now}})$。
6. 做梯度下降更新 DQN 的参数：
$$\boldsymbol{w}_{\text{new}} \;\leftarrow\; \boldsymbol{w}_{\text{now}} \;-\; \alpha \cdot \delta_j \cdot \nabla_{\boldsymbol{w}} Q(s_j, a_j; \boldsymbol{w}_{\text{now}}).$$
7. 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：
$$\boldsymbol{w}_{\text{new}}^{-} \;\leftarrow\; \tau \cdot \boldsymbol{w}_{\text{new}} \;+\; (1-\tau) \cdot \boldsymbol{w}_{\text{now}}^{-}.$$

6.2.6 总结

本节研究了 DQN 的高估问题以及解决方案。DQN 的高估不是 DQN 模型造成的，不是 DQN 的本质属性；高估只是因为原始 Q 学习算法不好。Q 学习算法产生高估的原因有两个：第一，自举导致偏差从一个 (s, a) 二元组传播到更多的二元组；第二，最大化造成 TD 目标高估真实价值。

想要解决高估问题，就要从自举、最大化这两方面下手。本节介绍了两种缓解高估的思路：使用目标网络、双 Q 学习。Q 学习算法与目标网络的结合可以缓解自举造成的偏差。双 Q 学习基于目标网络的想法，进一步将 TD 目标的计算分解成选择和求值两步，缓解了最大化造成的高估。图 6.5 总结了本节研究的三种算法。

	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

图 6.5: 三种 TD 算法的对比。

注 如果使用原始 Q 学习算法，自举和最大化的麻烦都会出现。在实践中，应当尽量使用双 Q 学习，它是三种算法中最好的。

注 如果使用 SARSA 算法（比如在 Actor-Critic 方法中），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。对于 SARSA，只需要解决自举问题，所以应当将目标网络应用到 SARSA。

6.3 对决网络 (Dueling Network)

本节介绍对决网络 (Dueling Network)，它是对 DQN 的神经网络的结构的改进。它的基本想法是将最优动作价值 $Q_\star$ 分解成最优状态价值 $V_\star$ 与最优优势 $D_\star$。对决网络的训练与 DQN 完全相同，可以用 Q 学习算法或者双 Q 学习算法。

6.3.1 最优优势函数

在介绍对决网络 (Dueling Network) 之前，先复习一些基础知识。动作价值函数 $Q_\pi(s,a)$ 是回报的期望：

$$Q_\pi(s,a) \;=\; \mathbb{E}\Big[U_t \,\Big|\, S_t=s, A_t=a\Big].$$

最优动作价值 $Q_\star$ 的定义是：

$$Q_\star(s,a) \;=\; \max_\pi Q_\pi(s,a), \qquad \forall\, s\in\mathcal{S},\; a\in\mathcal{A}.$$

状态价值函数 $V_\pi(s)$ 是 $Q_\pi(s,a)$ 关于 a 的期望：

$$V_\pi(s) \;=\; \mathbb{E}_{A\sim\pi}\Big[Q_\pi(s,A)\Big].$$

最优状态价值函数 $V_\star$ 的定义是：

$$V_\star \;=\; \max_\pi V_\pi(s), \qquad \forall\, s\in\mathcal{S}.$$

最优优势函数 (Optimal Advantage Function) 的定义是：

$$\boxed{D_\star(s,a) \;\triangleq\; Q_\star(s,a) \;-\; V_\star(s).}$$

通过数学推导，可以证明下面的定理：

定理 6.1

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

♡

6.3.2 对决网络

与 DQN 一样，对决网络 (Dueling Network) 也是对最优动作价值函数 $Q_\star$ 的近似。对决网络与 DQN 的区别在于神经网络结构不同。由于对决网络与 DQN 都是对 $Q_\star$ 的近似，可以用完全相同的算法训练两种神经网络。

对决网络由两个神经网络组成。一个神经网络记作 $D(s,a;\boldsymbol{w}^D)$，它是对最优优势函数 $D_\star(s,a)$ 的近似。另一个神经网络记作 $V(s,a;\boldsymbol{w}^V)$，它是对最优状态价值函数 $V_\star(s,a)$ 的近似。把定理 6.1 中的 $D_\star$ 和 $V_\star$ 替换成相应的神经网络，那么最优动作价值函数 $Q_\star$ 就被近似成下面的神经网络：

$$\boxed{Q(s,a;\boldsymbol{w}) \;\triangleq\; V(s;\boldsymbol{w}^V) \;+\; D(s,a;\boldsymbol{w}^D) \;-\; \max_{a\in\mathcal{A}} D(s,a;\boldsymbol{w}^D).} \tag{6.1}$$

公式左边的 $Q(s,a;\boldsymbol{w})$ 就是对决网络，它是对最优动作价值函数 $Q_\star$ 的近似。它的参数记作 $\boldsymbol{w} \triangleq (\boldsymbol{w}^V;\boldsymbol{w}^D)$。

对决网络的结构如图 6.6 所示。可以让两个神经网络 $D(s,a;\boldsymbol{w}^D)$ 与 $V(s,a;\boldsymbol{w}^V)$ 共享部分卷积层；这些卷积层把输入的状态 s 映射成特征向量，特征向量是“优势头”与“状态价值头”的输入。优势头输出一个向量，向量的维度是动作空间的大小 $|\mathcal{A}|$，向量每个元素对应一个动作。举个例子，动作空间是 $\mathcal{A}=\{左,右,上\}$，优势头的输出是三个值：

$$D\big(s,\,左;\,\boldsymbol{w}^D\big) \;=\; -90, \qquad D\big(s,\,右;\,\boldsymbol{w}^D\big) \;=\; -420, \qquad D\big(s,\,上;\,\boldsymbol{w}^D\big) \;=\; 30.$$

状态价值头输出的是一个实数，比如

$$V\big(s;\,\boldsymbol{w}^V\big) \;=\; 300.$$

首先计算

$$\max_a D\big(s,a;\boldsymbol{w}^D\big) \;=\; \max\big\{-90,-420,30\big\} \;=\; 30.$$

然后用公式 (6.1) 计算出：

$$Q\big(s,\,左;\,\boldsymbol{w}\big) \;=\; 180, \qquad Q\big(s,\,右;\,\boldsymbol{w}\big) \;=\; -150, \qquad Q\big(s,\,上;\,\boldsymbol{w}\big) \;=\; 300.$$

这样就得到了对决网络的最终输出。

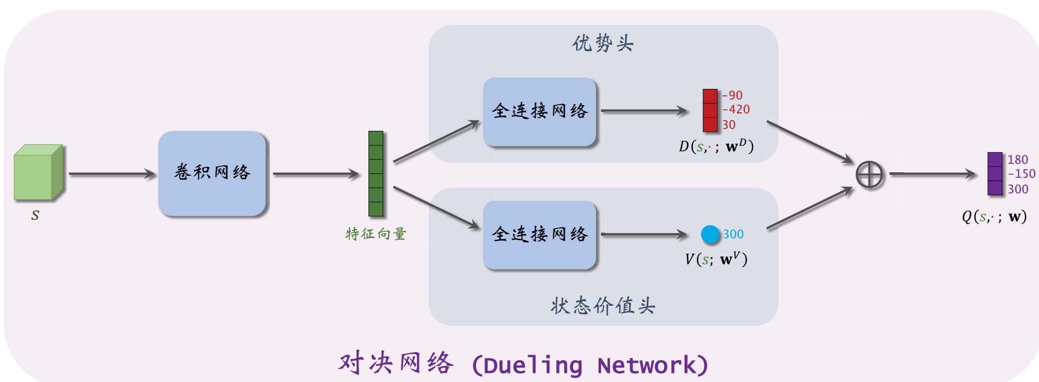


图 6.6: 对决网络的结构。输入是状态 s；红色的向量是每个动作的优势值；蓝色的标量是状态价值；最终输出的紫色向量是每个动作的动作价值。

6.3.3 解决不唯一性

读者可能会有下面的疑问。对决网络是由定理 6.1 推导出的，而定理中最右的一项恒等于零：

$$\max_{a\in\mathcal{A}} D_\star\big(s,a\big) \;=\; 0, \qquad \forall\, s\in\mathcal{S}.$$

也就是说，可以把最优动作价值写成两种等价形式：

$$\begin{aligned} Q_\star\big(s,a\big) \;&=\; V_\star\big(s\big)+D_\star\big(s,a\big) && \text{（第一种形式）}\\ &=\; V_\star\big(s\big)+D_\star\big(s,a\big)-\max_{a\in\mathcal{A}} D_\star\big(s,a\big). && \text{（第二种形式）}\end{aligned}$$

之前我们根据第二种形式实现对决网络。我们可否根据第一种形式，把对决网络按照下面的方式实现呢：

$$Q(s,a;\,\boldsymbol{w}) \;=\; V(s;\,\boldsymbol{w}^V) + D(s,a;\,\boldsymbol{w}^D)?$$

答案是不可以这样实现对决网络，因为这样会导致不唯一性。假如这样实现对决网络，那么 V 和 D 可以随意上下波动，比如一个增大 100，另一个减小 100：

$$\begin{aligned} V(s;\,\tilde{\boldsymbol{w}}^V) &\;\triangleq\; V(s;\,\boldsymbol{w}^V) + 100, \\ D(s,a;\,\tilde{\boldsymbol{w}}^D) &\;\triangleq\; D(s,a;\,\boldsymbol{w}^D) - 100. \end{aligned}$$

这样的上下波动不影响最终的输出：

$$V(s;\,\boldsymbol{w}^V) + D(s,a;\,\boldsymbol{w}^D) \;=\; V(s;\,\tilde{\boldsymbol{w}}^V) + D(s,a;\,\tilde{\boldsymbol{w}}^D).$$

这就意味着 V 和 D 的参数可以很随意地变化，却不会影响输出的 Q。我们不希望这种情况出现，因为这会导致训练的过程中参数不稳定。

因此很有必要在对决网络中加入 $\max_{a\in\mathcal{A}} D(s,a;\boldsymbol{w}^D)$ 这一项。它使得 V 和 D 不能随意上下波动。假如让 V 变大 100，让 D 变小 100，则对决网络的输出会增大 100，而非不变：

$$\begin{aligned} & V(s;\,\tilde{\boldsymbol{w}}^V) \;+\; D(s,a;\,\tilde{\boldsymbol{w}}^D) \;-\; \max_a D(s,a;\,\tilde{\boldsymbol{w}}^D) \\ =\; & V(s;\,\boldsymbol{w}^V) \;+\; D(s,a;\,\boldsymbol{w}^D) \;-\; \max_a D(s,a;\,\boldsymbol{w}^D) + 100. \end{aligned}$$

以上讨论说明了为什么 $\max_{a\in\mathcal{A}} D(s,a;\boldsymbol{w}^D)$ 这一项不能省略。

6.3.4 对决网络的实际实现

按照定理 6.1，对决网络应该定义成：

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

最右边的 max 项的目的是解决不唯一性。实际实现的时候，用 mean 代替 max 会有更好的效果。所以实际上会这样定义对决网络：

$$\boxed{Q(s,a;\,\boldsymbol{w}) \;\triangleq\; V(s;\,\boldsymbol{w}^V) \;+\; D(s,a;\,\boldsymbol{w}^D) \;-\; \operatorname*{mean}_{a\in\mathcal{A}} D(s,a;\,\boldsymbol{w}^D).}$$

对决网络与 DQN 都是对最优动作价值函数 $Q_\star$ 的近似，所以对决网络与 DQN 的训练和决策是完全一样的。比如可以这样训练对决网络：

- 用 ϵ-greedy 算法控制智能体，收集经验，把 (s_j,a_j,r_j,s_{j+1}) 这样的四元组存入经验回放数组。
- 从数组里随机抽取四元组，用双 Q 学习算法更新对决网络参数 $\boldsymbol{w}=(\boldsymbol{w}^D,\boldsymbol{w}^V)$。

完成训练之后，基于当前状态 s_t，让对决网络给所有动作打分，然后选择分数最高的动作：

$$a_t \;=\; \operatorname*{argmax}_{a\in\mathcal{A}}\; Q(s_t,a;\,\boldsymbol{w}).$$

简而言之，怎么样训练 DQN，就怎么样训练对决网络；怎么样用 DQN 做控制，就怎么

样用对决网络做控制。如果一个技巧能改进 DQN 的训练，这个技巧也能改进对决网络。同样的道理，Q 学习算法导致 DQN 出现高估，同样也会导致对决网络出现高估。

6.4 噪声网络

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

6.4.1 噪声网络的原理

把神经网络中的参数 $\boldsymbol{w}$ 替换成 $\boldsymbol{\mu}+\boldsymbol{\sigma}\circ\boldsymbol{\xi}$。此处的 $\boldsymbol{\mu}$、$\boldsymbol{\sigma}$、$\boldsymbol{\xi}$ 的形状与 $\boldsymbol{w}$ 完全相同。$\boldsymbol{\mu}$、$\boldsymbol{\sigma}$ 分别表示均值和标准差，它们是神经网络的参数，需要从经验中学习。$\boldsymbol{\xi}$ 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。符号 “$\circ$” 表示逐项乘积。如果 $\boldsymbol{w}$ 是向量，那么有

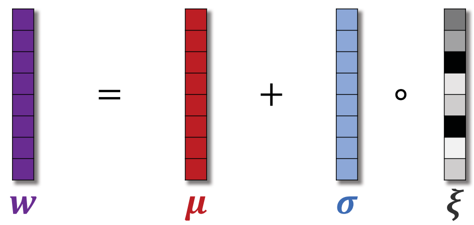


图 6.7: 这个例子中，$\boldsymbol{w}$、$\boldsymbol{\mu}$、$\boldsymbol{\sigma}$、$\boldsymbol{\xi}$ 是形状相同的向量。

$$w_i \;=\; \mu_i \,+\, \sigma_i \cdot \xi_i.$$

如果 w 是矩阵，那么有

$$w_{ij} \;=\; \mu_{ij} \,+\, \sigma_{ij} \cdot \xi_{ij}.$$

噪声网络的意思是参数 $\boldsymbol{w}$ 的每个元素 w_i 从均值为 μ_i、标准差为 σ_i 的正态分布中抽取。

举个例子，某一个全连接层记作：

$$\boldsymbol{z} \;=\; \mathrm{ReLU}\big(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}\big).$$

公式中的向量 $\boldsymbol{x}$ 是输入，矩阵 $\boldsymbol{W}$ 和向量 $\boldsymbol{b}$ 是参数，ReLU 是激活函数，$\boldsymbol{z}$ 是这一层的输出。噪声网络把这个全连接层替换成：

$$\boldsymbol{z} \;=\; \mathrm{ReLU}\Big(\big(\boldsymbol{W}^{\mu}+\boldsymbol{W}^{\sigma}\circ\boldsymbol{W}^{\xi}\big)\boldsymbol{x} \,+\, \big(\boldsymbol{b}^{\mu}+\boldsymbol{b}^{\sigma}\circ\boldsymbol{b}^{\xi}\big)\Big).$$

公式中的 $\boldsymbol{W}^{\mu}$、$\boldsymbol{W}^{\sigma}$、$\boldsymbol{b}^{\mu}$、$\boldsymbol{b}^{\sigma}$ 是参数，需要从经验中学习。矩阵 $\boldsymbol{W}^{\xi}$ 和向量 $\boldsymbol{b}^{\xi}$ 的每个元素都是独立从 $\mathcal{N}(0,1)$ 中随机抽取的，表示噪声。

训练噪声网络的方法与训练标准的神经网络完全相同，都是做反向传播计算梯度，然后用梯度更新神经参数。把损失函数记作 L。已知梯度 $\frac{\partial L}{\partial \boldsymbol{z}}$，可以用链式法则算出损失关于参数的梯度：

$$\frac{\partial L}{\partial \boldsymbol{W}^{\mu}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}^{\mu}} \cdot \frac{\partial L}{\partial \boldsymbol{z}}, \qquad \frac{\partial L}{\partial \boldsymbol{b}^{\mu}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}^{\mu}} \cdot \frac{\partial L}{\partial \boldsymbol{z}},$$
$$\frac{\partial L}{\partial \boldsymbol{W}^{\sigma}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}^{\sigma}} \cdot \frac{\partial L}{\partial \boldsymbol{z}}, \qquad \frac{\partial L}{\partial \boldsymbol{b}^{\sigma}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}^{\sigma}} \cdot \frac{\partial L}{\partial \boldsymbol{z}}.$$

然后可以做梯度下降更新参数 $\boldsymbol{W}^{\mu}$、$\boldsymbol{W}^{\sigma}$、$\boldsymbol{b}^{\mu}$、$\boldsymbol{b}^{\sigma}$。

6.4.2 噪声 DQN

噪声网络可以用于 DQN。标准的 DQN 记作 $Q(s,a;\boldsymbol{w})$，其中的 $\boldsymbol{w}$ 表示参数。把 $\boldsymbol{w}$ 替换成 $\boldsymbol{\mu}+\boldsymbol{\sigma}\circ\boldsymbol{\xi}$，得到噪声 DQN，记作：

$$\widetilde{Q}\big(s,a,\boldsymbol{\xi};\,\boldsymbol{\mu},\boldsymbol{\sigma}\big)\;\triangleq\;Q\big(s,a;\,\boldsymbol{\mu}+\boldsymbol{\sigma}\circ\boldsymbol{\xi}\big).$$

其中的 $\boldsymbol{\mu}$ 和 $\boldsymbol{\sigma}$ 是参数，一开始随机初始化，然后从经验中学习；而 $\boldsymbol{\xi}$ 则是随机生成，每个元素都从 $\mathcal{N}(0,1)$ 中抽取。噪声 DQN 的参数数量比标准 DQN 多一倍。

收集经验： DQN 属于异策略 (Off-policy)。我们用任意的行为策略 (Bahavior Policy) 控制智能体，收集经验，事后做经验回放更新参数。在之前章节中，我们用 ϵ-Greedy 作为行为策略：

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

ϵ-Greedy 策略带有一定的随机性，可以让智能体尝试更多动作，探索更多状态。

噪声 DQN 本身就带有随机性，可以鼓励探索，起到 ϵ-Greedy 策略相同的作用。我们直接用

$$a_t\;=\;\underset{a\in\mathcal{A}}{\operatorname{argmax}}\;\widetilde{Q}\big(s,a,\boldsymbol{\xi};\,\boldsymbol{\mu},\boldsymbol{\sigma}\big)$$

作为行为策略，效果比 ϵ-Greedy 更好。每做一个决策，要重新随机生成一个 $\boldsymbol{\xi}$。

Q 学习算法： 训练的时候，每一轮从经验回放数组中随机抽样出一个四元组，记作 (s_j,a_j,r_j,s_{j+1})。从标准正态分布中做抽样，得到 $\boldsymbol{\xi}'$ 的每一个元素。计算 TD 目标：

$$\widehat{y}_j\;=\;r_j\;+\;\gamma\cdot\max_{a\in\mathcal{A}}\widetilde{Q}\big(s_{j+1},a,\boldsymbol{\xi}';\,\boldsymbol{\mu},\boldsymbol{\sigma}\big).$$

把损失函数记作：

$$L\big(\boldsymbol{\mu},\boldsymbol{\sigma}\big)\;=\;\frac{1}{2}\Big[\widetilde{Q}\big(s_j,a_j,\boldsymbol{\xi};\,\boldsymbol{\mu},\boldsymbol{\sigma}\big)\;-\;\widehat{y}_j\Big]^2,$$

其中的 $\boldsymbol{\xi}$ 也是随机生成的噪声，但是它与 $\boldsymbol{\xi}'$ 不同。然后做梯度下降更新参数：

$$\boldsymbol{\mu}\;\leftarrow\;\boldsymbol{\mu}-\alpha_{\mu}\cdot\nabla_{\boldsymbol{\mu}}L\big(\boldsymbol{\mu},\boldsymbol{\sigma}\big),\qquad\quad\boldsymbol{\sigma}\;\leftarrow\;\boldsymbol{\sigma}-\alpha_{\sigma}\cdot\nabla_{\boldsymbol{\sigma}}L\big(\boldsymbol{\mu},\boldsymbol{\sigma}\big).$$

公式中的 α_μ 和 α_σ 是学习率。这样做梯度下降更新参数，可以让损失函数减小，让噪声 DQN 的预测更接近 TD 目标。

做决策： 做完训练之后，可以用噪声 DQN 做决策。做决策的时候不再需要噪声，因此可以把参数 $\boldsymbol{\sigma}$ 设置成全零，只保留参数 $\boldsymbol{\mu}$。这样一来，噪声 DQN 就变成标准的 DQN：

$$\underbrace{\widetilde{Q}\big(s,a,\boldsymbol{\xi}';\,\boldsymbol{\mu},\mathbf{0}\big)}_{\text{噪声 DQN}}\;=\;\underbrace{Q\big(s,a;\boldsymbol{\mu}\big)}_{\text{标准 DQN}}.$$

在训练的时候往 DQN 的参数中加入噪声，不仅有利于探索，还能增强鲁棒性。鲁棒性的意思是即使参数被扰动，DQN 也能对动作价值 $Q_\star$ 做出可靠的估计。为什么噪声可以让 DQN 有更强的鲁棒性呢？

假设在训练的过程中不加入噪声。把学出的参数记作 $\boldsymbol{\mu}$。当参数严格等于 $\boldsymbol{\mu}$ 的时候，DQN 可以对最优动作价值做出较为准确的估计。但是对 $\boldsymbol{\mu}$ 做较小的扰动，就可能会让

DQN 的输出偏离很远。所谓“失之毫厘，谬以千里”。

噪声 DQN 训练的过程中，参数带有噪声：$\boldsymbol{w}=\boldsymbol{\mu}+\boldsymbol{\sigma}\circ\boldsymbol{\xi}$。训练迫使 DQN 在参数带噪声的情况下最小化 TD 误差，也就是迫使 DQN 容忍对参数的扰动。训练出的 DQN 具有鲁棒性：参数不严格等于 $\boldsymbol{\mu}$ 也没关系，只要参数在 $\boldsymbol{\mu}$ 的邻域内，DQN 做出的预测都应该比较合理。用噪声 DQN，不会出现“失之毫厘，谬以千里”。

6.4.3 训练流程

实际编程实现 DQN 的时候，应该将本章的四种技巧——优先经验回放、双 Q 学习、对决网络、噪声 DQN——全部用到。应该用**对决网络**的神经网络结构，而不是简单的 DQN 结构。往对决网络中的参数 $\boldsymbol{w}$ 中加入噪声，得到噪声 DQN，记作 $\widetilde{Q}(s,a,\boldsymbol{\xi};\,\boldsymbol{\mu},\boldsymbol{\sigma})$。训练要用**双 Q 学习**、**优先经验回放**，而不是原始的 Q 学习。双 Q 学习需要目标网络 $\widetilde{Q}(s,a,\boldsymbol{\xi};\,\boldsymbol{\mu}^{-},\boldsymbol{\sigma}^{-})$ 计算 TD 目标。它跟噪声 DQN 的结构相同，但是参数不同。

初始的时候，随机初始化 $\boldsymbol{\mu}$、$\boldsymbol{\sigma}$，并且把它们赋值给目标网络参数：$\boldsymbol{\mu}^{-}\leftarrow\boldsymbol{\mu}$、$\boldsymbol{\sigma}^{-}\leftarrow\boldsymbol{\sigma}$。然后重复下面的步骤更新参数。把当前的参数记作 $\boldsymbol{\mu}_{\text{now}}$、$\boldsymbol{\sigma}_{\text{now}}$、$\boldsymbol{\mu}_{\text{now}}^{-}$、$\boldsymbol{\sigma}_{\text{now}}^{-}$。

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 $\boldsymbol{\xi}$。对噪声 DQN 做正向传播，得到：
$$\widehat{q}_j \;=\; \widetilde{Q}\big(s_j,a_j,\,\boldsymbol{\xi};\,\boldsymbol{\mu}_{\text{now}},\boldsymbol{\sigma}_{\text{now}}\big).$$
3. 根据噪声 DQN 选出最优动作：
$$\tilde{a}_{j+1} \;=\; \underset{a\in\mathcal{A}}{\operatorname{argmax}}\ \widetilde{Q}\big(s_{j+1},a,\,\boldsymbol{\xi};\,\boldsymbol{\mu}_{\text{now}},\boldsymbol{\sigma}_{\text{now}}\big).$$
4. 用标准正态分布生成 $\boldsymbol{\xi}'$。根据目标网络计算价值：
$$\widehat{q_{j+1}^{-}} \;=\; \widetilde{Q}\big(s_{j+1},\tilde{a}_{j+1},\,\boldsymbol{\xi}';\,\boldsymbol{\mu}_{\text{now}}^{-},\boldsymbol{\sigma}_{\text{now}}^{-}\big).$$
5. 计算 TD 目标和 TD 误差：
$$\widehat{y_j^{-}} \;=\; r_j+\gamma\cdot\widehat{q_{j+1}^{-}} \qquad \text{和} \qquad \delta_j \;=\; \widehat{q}_j-\widehat{y_j^{-}}.$$
6. 设 α_{μ} 和 α_{σ} 为学习率。做梯度下降更新噪声 DQN 的参数：
$$\begin{aligned}\boldsymbol{\mu}_{\text{new}} &\;\leftarrow\; \boldsymbol{\mu}_{\text{now}}-\alpha_{\mu}\cdot\delta_j\cdot\nabla_{\boldsymbol{\mu}}\,\widetilde{Q}\big(s_j,a_j,\,\boldsymbol{\xi};\,\boldsymbol{\mu}_{\text{now}},\boldsymbol{\sigma}_{\text{now}}\big),\\ \boldsymbol{\sigma}_{\text{new}} &\;\leftarrow\; \boldsymbol{\sigma}_{\text{now}}-\alpha_{\sigma}\cdot\delta_j\cdot\nabla_{\boldsymbol{\sigma}}\,\widetilde{Q}\big(s_j,a_j,\,\boldsymbol{\xi};\,\boldsymbol{\mu}_{\text{now}},\boldsymbol{\sigma}_{\text{now}}\big).\end{aligned}$$
7. 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：
$$\begin{aligned}\boldsymbol{\mu}_{\text{new}}^{-} &\;\leftarrow\; \tau\cdot\boldsymbol{\mu}_{\text{new}}\;+\;\big(1-\tau\big)\cdot\boldsymbol{\mu}_{\text{now}}^{-},\\ \boldsymbol{\sigma}_{\text{new}}^{-} &\;\leftarrow\; \tau\cdot\boldsymbol{\sigma}_{\text{new}}\;+\;\big(1-\tau\big)\cdot\boldsymbol{\sigma}_{\text{now}}^{-}.\end{aligned}$$

第六章 相关文献

训练 DQN 用到的经验回放是由 Lin 在 1993 年的博士论文 [63] 中提出的。优先经验回放是由 Schaul 等人 2015 年的论文 [87] 提出。目标网络由 Mnih 等人 2015 年的论文 [71] 提出。双 Q 学习由 van Hasselt 2010 年的论文 [108] 提出。双 Q 学习与 DQN 的结合被称为 Double DQN，由 van Hasselt 等人 2010 年的论文提出 [109]。对决网络在 Wang 等人 2016 年的论文中提出 [114]。噪声网络在 Fortunato 等人 2018 年的论文中提出 [40]。

Hessel 等人在 2018 年发表的论文 [47] 将优先经验回放、双 Q 学习、对决网络、多步 TD 目标等方法结合，改进 DQN，把组成称为 Rainbow，用实验证明高级技巧的有效性。此外，Rainbow 还用到了 Distributional Learning [12]，这种技巧也非常有用。