

第四章 DQN 与 Q 学习

本章的内容是价值学习的基础。第 4.1 用神经网络近似最优动作价值函数 $Q^*(s, a)$ ，把这个神经网络称为深度 Q 网络 (DQN)。本章内容的难点在于训练 DQN 所用的时间差分算法 (TD)。第 4.2 节以“驾车时间估计”类比 DQN，讲解 TD 算法第 4.3 节推导训练 DQN 用的 Q 学习算法（一种 TD 算法）。第 4.4 节介绍表格形式的 Q 学习算法。第 4.5 节解释同策略 (On-policy) 与异策略 (Off-policy) 的区别；本章介绍的 Q 学习算法属于异策略。

4.1 DQN

在学习 DQN 之前，首先复习一些基础知识。在一局 (Episode) 游戏中，把从起始到结束的所有奖励记作：

$$R_1, \dots, R_t, \dots, R_n.$$

定义折扣率 $\gamma \in [0, 1]$ 。折扣回报的定义是：

$$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \dots + \gamma^{n-t} \cdot R_n.$$

在游戏尚未结束的 t 时刻， U_t 是一个未知的随机变量，其中的随机性来自于 t 时刻之后的所有状态与动作。动作价值函数的定义是：

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

公式中的期望消除了 t 时刻之后的所有状态 S_{t+1}, \dots, S_n 与所有动作 A_{t+1}, \dots, A_n 。最优动作价值函数用最大化消除策略 π ：

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

可以这样理解 Q_* ：已知 s_t 和 a_t ，不论未来采取什么样的策略 π ，回报 U_t 的期望不可能超过 Q_* 。

最优动作价值函数的用途：假如我们知道 Q_* ，我们就能用它做控制。举个例子，超级玛丽游戏中的动作空间是 $\mathcal{A} = \{\text{左, 右, 上}\}$ 。给定当前状态 s_t ，智能体该执行哪个动作呢？假设我们已知 Q_* 函数，那么我们就让 Q_* 给三个动作打分，比如：

$$Q_*(s_t, \text{左}) = 370, \quad Q_*(s_t, \text{右}) = -21, \quad Q_*(s_t, \text{上}) = 610.$$

这三个值是什么意思呢？ $Q_*(s_t, \text{左}) = 130$ 的意思是：如果现在智能体选择向左走，不论之后采取什么策略 π ，那么回报 U_t 的期望最多不会超过 370。同理，其他两个最优动作价值的也是回报的期望的上界。根据 Q_* 的评分，智能体应该选择向上跳，因为这样可以最大化回报 U_t 的期望。

我们希望知道 Q_* ，因为它就像是先知一般，可以预见未来，在 t 时刻就预见 t 到 n 时刻之间的累计奖励的期望。假如我们有 Q_* 这位先知，我们就遵照按照先知的指导，最大化未来的累计奖励。然而在实践中我们无法得到 Q_* 的函数表达式。是否有可能近似

出 Q_* 这位先知呢？对于超级玛丽这样的游戏，学出来一个“先知”并不难。假如让我重复玩超级玛丽一亿次，那我就像是先知一样：告诉我当前状态，我能准确判断出当前最优的动作是什么。这说明只要有足够多的“经验”，就能训练出超级玛丽中的“先知”。

最优动作价值函数的近似：在实践中，近似学习“先知” Q_* 最有效的办法是深度 Q 网络 (Deep Q Network)，缩写是 DQN，记作 $Q(s, a; \mathbf{w})$ ，其结构在图 4.1 中描述。其中的 \mathbf{w} 表示神经网络中的参数；一开始随机初始化 \mathbf{w} ，随后用“经验”去学习 \mathbf{w} 。学习的目标是：对于所有的 s 和 a ，DQN 的预测 $Q(s, a; \mathbf{w})$ 尽量接近 $Q_*(s, a)$ 。之后几节的内容都是如何学习 \mathbf{w} 。

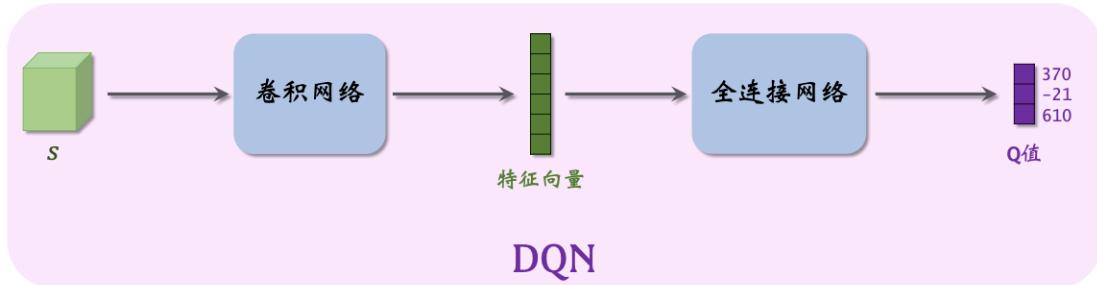


图 4.1: DQN 的神经网络结构。输入是状态 s ；输出是每个动作的 Q 值。。

可以这样理解 DQN 的表达式 $Q(s, a; \mathbf{w})$ 。DQN 的输出是离散动作空间 \mathcal{A} 上的每个动作的 Q 值（即给每个动作的评分，分数越高，动作越好）。举个例子，动作空间是 $\mathcal{A} = \{\text{左, 右, 上}\}$ ，那么动作空间的大小等于 $|\mathcal{A}| = 3$ ，DQN 的输出是 3 维的向量 $\hat{\mathbf{q}}$ ，向量每个元素对应一个动作。在图 4.1 中，DQN 的输出是

$$\begin{aligned}\hat{q}_1 &= Q(s, \text{左}; \mathbf{w}) = 370, \\ \hat{q}_2 &= Q(s, \text{右}; \mathbf{w}) = -21, \\ \hat{q}_3 &= Q(s, \text{上}; \mathbf{w}) = 610.\end{aligned}$$

总结一下，DQN 的输出是 $|\mathcal{A}|$ 维的向量 $\hat{\mathbf{q}}$ ，包含所有动作的价值。而我们常用的符号 $Q(s, a; \mathbf{w})$ 是标量，是动作 a 对应的动作价值。可以把动作 a 看做是一个序号，比如 $a = \text{“右”}$ 是 \mathcal{A} 中第二号动作，那么 $Q(s, a; \mathbf{w})$ 就是向量 $\hat{\mathbf{q}}$ 的第二号元素 $\hat{q}_2 = -21$ 。

DQN 的梯度：在训练 DQN 的时候，需要对 DQN 关于神经网络参数 \mathbf{w} 求梯度。用

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

表示函数值 $Q(s, a; \mathbf{w})$ 关于参数 \mathbf{w} 的梯度。因为函数值 $Q(s, a; \mathbf{w})$ 是一个实数，所以梯度的形状与 \mathbf{w} 完全相同：如果 \mathbf{w} 是 $d \times 1$ 的向量，那么梯度也是 $d \times 1$ 的向量；如果 \mathbf{w} 是 $d_1 \times d_2$ 的矩阵，那么梯度也是 $d_1 \times d_2$ 的矩阵；如果 \mathbf{w} 是 $d_1 \times d_2 \times d_3$ 的张量，那么梯度也是 $d_1 \times d_2 \times d_3$ 的张量。

给定观测值 s 和 a （比如 $a = \text{“左”}$ ），可以用反向传播计算出梯度 $\nabla_{\mathbf{w}} Q(s, \text{“左”}; \mathbf{w})$ 。在编程实现的时候，TensorFlow 和 PyTorch 可以对 DQN 输出向量的一个元素（比如 $Q(s, \text{“左”}; \mathbf{w})$ ）关于变量 \mathbf{w} 自动求梯度，得到的梯度的形状与 \mathbf{w} 完全相同。

4.2 时间差分 (TD) 算法

训练 DQN 最常用的算法是时间差分 (Temporal Difference)，缩写 TD。TD 算法不太好理解，所以本节举一个通俗易懂的例子讲解 TD 算法。

4.2.1 驾车时间预测的例子

假设我们有一个模型 $Q(s, d; \mathbf{w})$ ，其中 s 是起点， d 是终点， \mathbf{w} 是参数，它可以预测出开车出行的时间开销。这个模型一开始不准确，甚至是纯随机的。但是随着很多人用这个模型，得到更多数据、更多训练，这个模型就会越来越准，会像谷歌地图一样准。

我们该如何训练这个模型呢？在用户出发前，用户告诉模型起点 s 和终点 d ，模型做一个预测 $\hat{q} = Q(s, d; \mathbf{w})$ 。当用户结束行程的时候，把实际驾车时间 y 反馈给模型。两者之差 $\hat{q} - y$ 反映出模型是高估还是低估了驾驶时间，可以以此来修正模型，让模型的估计更准确。

假设我是个用户，我要从北京驾车去上海。从北京出发之前，我让模型做预测，模型告诉我总车程是 14 小时。

$$\hat{q} \triangleq Q(\text{“北京”}, \text{“上海”}; \mathbf{w}) = 14.$$

当我到达上海，我知道自己在路上花的时间是 16 小时，并将结果反馈给模型；见图 4.2。



图 4.2：模型估计驾驶时间是 $Q(s, d; \mathbf{w}) = 14$ ，而实际花费时间 $y = 16$ 。

可以用在线梯度下降对模型做一次更新，具体做法如下。把我的这次旅程作为一组训练数据：

$$s = \text{“北京”}, \quad d = \text{“上海”}, \quad \hat{q} = 14, \quad y = 16.$$

我们希望估计值 $\hat{q} = Q(s, d; \mathbf{w})$ 尽量接近真实观测到的 y ，所以用两者的平方差作为损失函数：

$$L(\mathbf{w}) = \frac{1}{2} [Q(s, d; \mathbf{w}) - y]^2.$$

用链式法则计算损失函数的梯度，得到：

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = (\hat{q} - y) \cdot \nabla_{\mathbf{w}} Q(s, d; \mathbf{w}),$$

然后做一次梯度下降更新模型参数 \mathbf{w} ：

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \nabla_{\mathbf{w}} L(\mathbf{w}),$$

此处的 α 是学习率，需要手动调。在完成一次梯度下降之后，如果再让模型做一次预测，

那么模型的预测值

$$Q(\text{“北京”}, \text{“上海”}; \mathbf{w})$$

会比原先更接近 $y = 16$.

4.2.2 TD 算法

接着上文驾车时间的例子。出发前模型估计全程时间为 $\hat{q} = 14$ 小时；模型建议的路线会途径济南。我从北京出发，过了 $r = 4.5$ 小时，我到达济南。此时我再让模型做一次预测，模型告诉我

$$\hat{q}' \triangleq Q(\text{“济南”}, \text{“上海”}; \mathbf{w}) = 11.$$

见图 4.3 的描述。假如此时我的车坏了，必须要在济南修理，我不得不取消此次行程。我没有完成旅途，那么我的这组数据是否能帮助训练模型呢？其实是可以的，用到的算法不再是在线梯度下降，而是时间差分 (Temporal Difference)，缩写为 TD。



图 4.3: 紫色的数字 14 和 11 是模型的估计值；蓝色的数字 4.5 是实际观测值。

下面解释 TD 算法的原理。回顾一下我们已有的数据：模型估计从北京到上海一共需要 $\hat{q} = 14$ 小时，我实际用了 $r = 4.5$ 小时到达济南，模型估计还需要 $\hat{q}' = 11$ 小时从济南到上海。到达济南时，根据模型最新估计，整个旅程的总时间为：

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

TD 算法将 $\hat{y} = 15.5$ 称为 TD 目标 (TD Target)，它比最初的估计 $\hat{q} = 14$ 更可靠。最初的估计 $\hat{q} = 14$ 纯粹是估计的，没有任何事实的成分。TD 目标 $\hat{y} = 15.5$ 也是个估计，但其中有事实的成分：其中的 $r = 4.5$ 就是实际的观测。

基于以上讨论，我们认为 TD 目标 $\hat{y} = 15.5$ 比模型最初的估计值

$$\hat{q} = Q(\text{“北京”}, \text{“上海”}; \mathbf{w}) = 14$$

更可靠，所以可以用 \hat{y} 对模型做“修正”。我们希望估计值 \hat{q} 尽量接近真实值 \hat{y} ，所以用两者的平方差作为损失函数：

$$L(\mathbf{w}) = \frac{1}{2} [Q(\text{“北京”}, \text{“上海”}; \mathbf{w}) - \hat{y}]^2.$$

把 \hat{y} 看做常数，尽管它依赖于 w 。¹ 计算损失函数的梯度：

$$\nabla_w L(w) = \underbrace{(\hat{q} - \hat{y})}_{\text{记作 } \delta} \cdot \nabla_w Q(\text{“北京”}, \text{“上海”}; w),$$

此处的 $\delta = \hat{q} - \hat{y}$ 称作 **TD 误差** (TD Error)。做一次梯度下降更新模型参数 w ：

$$w \leftarrow w - \alpha \cdot \delta \cdot \nabla_w Q(\text{“北京”}, \text{“上海”}; w).$$

TD 算法指的是用此公式更新模型参数 w 。

如果你仍然不理解 TD 算法，那么请换个角度来思考问题。模型估计从北京到上海全程需要 $\hat{q} = 14$ 小时，模型还估计从济南到上海需要 $\hat{q}' = 11$ 小时。这就相当于模型做了这样的估计：从北京到济南需要的时间为

$$\hat{q} - \hat{q}' = 14 - 11 = 3.$$

而我真实花费 $r = 4.5$ 小时。模型的估计与我的真实观测之差为

$$\delta = 3 - 4.5 = -1.5.$$

这就是 TD 误差！以上分析说明 TD 误差 δ 就是模型估计与真实观测之差。TD 算法的目的是通过更新参数 w 使得目标函数 $L(w) = \frac{1}{2}\delta^2$ 减小。

¹根据定义，TD 目标 $\hat{y} = r + \hat{q}'$ ，而 $\hat{q}' = Q(\text{“济南”}, \text{“上海”}; w)$ 依赖于 w 。因此， \hat{y} 其实是 w 的函数。然而 TD 算法忽视这一点，在求梯度的时候，将 \hat{y} 视为常数，而非 w 的函数。

4.3 用 TD 训练 DQN

上一节以驾车时间预测为例介绍了 TD 算法。本节用 TD 算法训练 DQN。第 4.3.1 小节推导算法，第 4.3.2 详细描述训练 DQN 的流程。注意，本节推导的算法是最原始，实践中效果不佳。实际实现 DQN 的时候，应当使用第 6 章介绍的高级技巧。

4.3.1 算法推导

下面我们推导训练 DQN 的 TD 算法。² 回忆一下回报的定义： $U_t = \sum_{k=t}^n \gamma^{k-t} \cdot R_k$ 。由这个定义可得：

$$U_t = R_t + \gamma \cdot \underbrace{\sum_{k=t+1}^n \gamma^{k-t} \cdot R_k}_{= U_{t+1}}.$$

回忆一下，最优动作价值函数可以写成

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

从上面两个公式出发，经过一系列数学推导（见附录 A），可以得到下面的定理。这个定理是**最优贝尔曼方程 (Optimal Bellman Equations)** 的一种形式。

定理 4.1. 最优贝尔曼方程

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



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。期望是关于随机变量 S_{t+1} 求的，所以需要从环境得到 S_{t+1} 的一个观测值。当智能体执行动作 a_t 之后，环境通过状态转移函数 $p(s_{t+1}|s_t, a_t)$ 计算出新状态 s_{t+1} ，并将其反馈给智能体。奖励 R_t 最多只依赖于 S_t 、 A_t 、 S_{t+1} 。那么当我们观测到 s_t 、 a_t 、 s_{t+1} ，则奖励 R_t 也被观测到，记作 r_t 。有了这个四元组：

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

我们就有了贝尔曼方程右边期望的一个蒙特卡洛近似。可以把贝尔曼方程写作：

$$Q_*(s_t, a_t) \approx r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q_*(s_{t+1}, a). \quad (4.1)$$

这是不是很像驾驶时间预测问题？左边的 $Q_*(s_t, a_t)$ 就像是模型预测“北京到上海”的总时间， r_t 像是实际观测的“北京到济南”的时间， $\gamma \cdot \max_{a \in \mathcal{A}} Q_*(s_{t+1}, a)$ 相当于模型预测剩余路程“济南到上海”的时间。见图 4.4 中的类比。

²严格地讲，此处推导的是“Q 学习算法”，它属于 TD 算法的一种。本节就称其为 TD 算法；下一节再具体介绍 Q 学习算法。

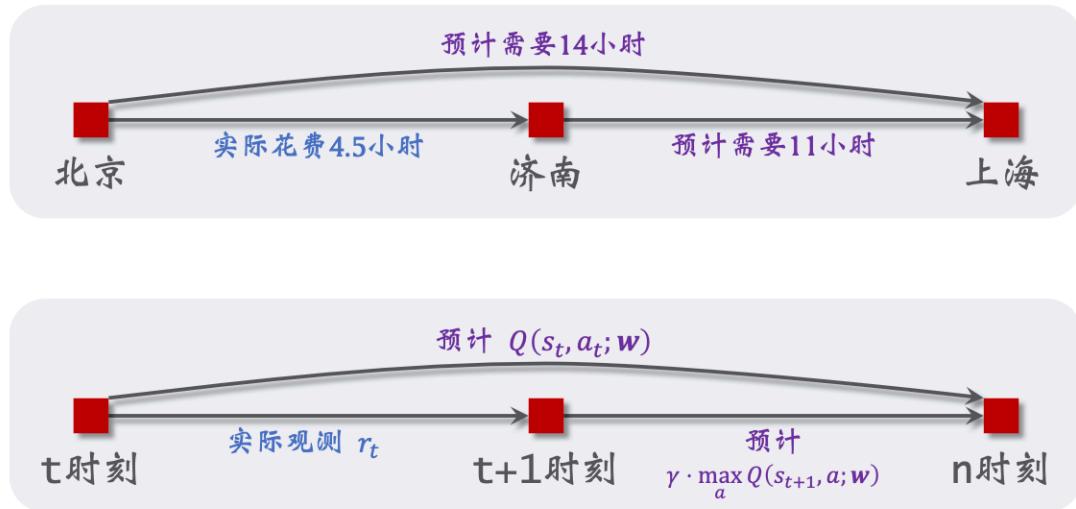


图 4.4: 用“驾车时间”类比 DQN。

把公式 4.1 中的最优动作价值函数 $Q_\star(s_t, a_t)$ 替换成神经网络 $Q(s_t, a_t; \mathbf{w})$, 得到:

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

左边的 $\hat{q}_t \triangleq Q(s_t, a_t; \mathbf{w})$ 是神经网络在 t 时刻做出的预测, 其中没有任何事实成分。右边的 TD 目标 \hat{y}_t 是神经网络在 $t+1$ 时刻做出的预测, 它部分基于真实观测到的奖励 r_t 。 \hat{q}_t 和 \hat{y}_t 两者都是对最优动作价值 $Q_\pi(s_t, a_t)$ 的估计, 但是 \hat{y}_t 部分基于事实, 因此更可信。应当鼓励 $\hat{q}_t \triangleq Q(s_t, a_t; \mathbf{w})$ 接近 \hat{y}_t 。定义损失函数:

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

假装 \hat{y} 是常数, 计算 L 关于 \mathbf{w} 的梯度:

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

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

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \delta_t \cdot \nabla_{\mathbf{w}} Q(s_t, a_t; \mathbf{w}).$$

这个公式就是训练 DQN 的 TD 算法。

4.3.2 训练流程

首先回顾上一节的结论。给定一个四元组 (s_t, a_t, r_t, s_{t+1}) , 我们可以计算出 DQN 的预测值

$$\hat{q}_t = Q(s_t, a_t; \mathbf{w}),$$

以及 TD 目标和 TD 误差:

$$\hat{y}_t = r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \mathbf{w}) \quad \text{和} \quad \delta_t = \hat{q}_t - \hat{y}_t.$$

TD 算法用这个公式更新 DQN 的参数：

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \delta_t \cdot \nabla_{\mathbf{w}} Q(s_t, a_t; \mathbf{w}).$$

注意，算法所需数据为 (s_t, a_t, r_t, s_{t+1}) 这个四元组，与控制智能体运动的策略 π 无关。这就意味着可以用任何策略控制智能体，同时记录下算法运动轨迹，作为 DQN 的训练数据。因此，DQN 的训练可以分割成两个独立的部分：收集训练数据、更新参数 \mathbf{w} 。

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：

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

把智能体在一局游戏中的轨迹记作：

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

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 \mathbf{w} ：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 当前的参数为 \mathbf{w}_{now} ，执行下面的步骤对参数做一次更新，得到新的参数 \mathbf{w}_{new} 。

1. 对 DQN 做正向传播，得到 Q 值：

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

2. 计算 TD 目标和 TD 误差：

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

3. 对 DQN 做反向传播，得到梯度：

$$\mathbf{g}_j = \nabla_{\mathbf{w}} Q(s_j, a_j; \mathbf{w}_{\text{now}}).$$

4. 做梯度下降更新 DQN 的参数：

$$\mathbf{w}_{\text{new}} \leftarrow \mathbf{w}_{\text{now}} - \alpha \cdot \delta_j \cdot \mathbf{g}_j.$$

智能体收集数据、更新 DQN 参数这两者可以同时进行。可以在智能体每执行一个动作之前（或之后），对 \mathbf{w} 做几次更新。

4.4 Q 学习算法

上一节用 TD 算法训练 DQN；准确地说，我们用的 TD 算法叫做 Q 学习算法 (Q-learning)。TD 算法是一大类算法，常见的有 Q 学习和 SARSA。Q 学习的目是学到最优动作价值函数 Q_* ；而 SARSA 的目的是学习动作价值函数 Q_π 。后面一章会介绍 SARSA 算法。

Q 学习是在 1989 年提出的，而 DQN 则是 2013 年才提出。从 DQN 的名字（深度 Q 网络）就能看出 DQN 与 Q 学习的联系。最初的 Q 学习都是以表格形式出现的。虽然表格形式的 Q 学习在实践不常用，还是建议读者有所了解。

用表格表示 Q_* ：假设状态空间 \mathcal{S} 和动作空间 \mathcal{A} 都是有限集，即集合中元素数量有限。³ 比如， \mathcal{S} 中一共有 3 种状态， \mathcal{A} 中一共有 4 种动作。那么最优动作价值函数 $Q_*(s, a)$ 可以表示为一个 3×4 的表格，比如右边的表格。基于当前状态 s_t ，做决策时使用的公式

	第 1 种 动作	第 2 种 动作	第 3 种 动作	第 4 种 动作
第 1 种 状态	380	-95	20	173
第 2 种 状态	-7	64	-195	210
第 3 种 状态	152	72	413	-80

图 4.5：最优动作价值函数 Q_* 表示成表格形式。

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q_*(s_t, a)$$

的意思是找到 s_t 对应的行（3 行中的某一行），找到该行最大的价值，返回该元素对应的动作。举个例子，当前状态 s_t 是第 2 种状态，那么我们查看第二行，发现该行最大的价值是 210，对应第四种动作。那么应当执行的动作 a_t 就是第四种动作。

该如何通过智能体的轨迹来学习这样一个表格呢？用一个表格 \tilde{Q} 来近似 Q_* 。首先初始化 \tilde{Q} ，可以让它是全零的表格。然后用表格形式的 Q 学习算法更新 \tilde{Q} ，每次更新表格的一个元素。最终 \tilde{Q} 会收敛到 Q^* 。

算法推导：首先复习一下最优贝尔曼方程：

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

我们对方程左右两边做近似：

- 方程左边的 $Q_*(s_t, a_t)$ 可以近似成 $\tilde{Q}(s_t, a_t)$ 。 $\tilde{Q}(s_t, a_t)$ 是表格在 t 时刻对 $Q_*(s_t, a_t)$ 做出的估计。
- 方程右边的期望是关于下一时刻状态 S_{t+1} 求的。给定当前状态 s_t ，智能体执行动作 a_t ，环境会给出奖励 r_t 和新的状态 s_{t+1} 。用观测到的 r_t 和 s_{t+1} 对期望做蒙特卡洛近似，得到：

$$r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q_*(s_{t+1}, a). \quad (4.2)$$

³如果 \mathcal{A} 是有限集，而 \mathcal{S} 是无限集，那么我们可以用神经网络形式的 Q 学习，即上一节的 DQN。如果 \mathcal{A} 是无限集，则问题属于连续控制，应当使用连续控制的方法，见第 10 章。

- 进一步把公式 (4.2) 中的 Q_* 近似成 \tilde{Q} , 得到

$$\hat{y}_t \triangleq r_t + \gamma \cdot \max_{a \in \mathcal{A}} \tilde{Q}(s_{t+1}, a).$$

把它称作 TD 目标。它是表格在 $t+1$ 时刻对 $Q_*(s_t, a_t)$ 做出的估计。

$\tilde{Q}(s_t, a_t)$ 和 \hat{y}_t 都是对最优动作价值 $Q_*(s_t, a_t)$ 的估计。由于 \hat{y}_t 部分基于真实观测到的奖励 r_t , 我们认为 \hat{y}_t 是更可靠的估计, 所以鼓励 $\tilde{Q}(s_t, a_t)$ 更接近 \hat{y}_t 。更新表格 \tilde{Q} 中 (s_t, a_t) 位置上的元素:

$$\tilde{Q}(s_t, a_t) \leftarrow \tilde{Q}(s_t, a_t) - \alpha \cdot \tilde{Q}(s_t, a_t) + \alpha \cdot \hat{y}_t.$$

这样可以使得 $\tilde{Q}(s_t, a_t)$ 更接近 \hat{y}_t 。

收集训练数据: Q 学习更新 \tilde{Q} 的公式不依赖于具体的策略, 所以 Q 学习属于异策略 (Off-policy)。可以用任何的行为策略收集经验 (经验即四元组 $(s_t, a_t, s_{t+1}, a_{t+1})$), 事后用经验回放训练目标策略 (即最新的 \tilde{Q} 表格)。比较常用的行为策略是 ϵ -greedy:

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

把智能体与环境交互得到的轨迹划分成 (s_t, a_t, r_t, s_{t+1}) 这样的四元组, 存入经验回放数组。

经验回放更新表格 \tilde{Q} : 随机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}) 。设当前表格为 \tilde{Q}_{now} 。更新表格中 (s_j, a_j) 位置上的元素, 把更新之后的表格记作 \tilde{Q}_{new} 。

1. 把表格 \tilde{Q}_{now} 中第 (s_j, a_j) 位置上的元素记作:

$$\hat{q}_j = \tilde{Q}_{\text{now}}(s_j, a_j).$$

2. 查看表格 \tilde{Q}_{now} 的第 s_{j+1} 行, 把该行的最大值记作:

$$\hat{q}_{j+1} = \max_a \tilde{Q}_{\text{now}}(s_{j+1}, a).$$

3. 计算 TD 目标和 TD 误差:

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

4. 更新表格中 (s_j, a_j) 位置上的元素:

$$\tilde{Q}_{\text{new}}(s_j, a_j) \leftarrow \tilde{Q}_{\text{now}}(s_j, a_j) - \alpha \cdot \delta_j.$$

收集经验与更新表格 \tilde{Q} 可以同时进行。每当智能体执行一次动作, 我们可以用经验回放对 \tilde{Q} 做几次更新。

4.5 同策略 (On-policy) 与异策略 (Off-policy)

在强化学习中经常会遇到两个专业术语：同策略 (On-policy) 和异策略 (Off-policy)。为了解释同策略和异策略，我们要从行为策略 (Behavior Policy) 和目标策略 (Target Policy) 讲起。

在强化学习中，我们让智能体与环境交互，记录下观测到的状态、动作、奖励，用这些数据来学习一个策略函数。在这一过程中，控制智能体与环境交互的策略被称作行为策略。行为策略的作用是收集经验 (Experience)，即观测的环境、动作、奖励。

训练的目的是得到一个策略函数，在结束训练之后，用这个策略函数来控制智能体；这个策略函数就叫做目标策略。在本章中，目标策略是一个确定性的策略，即用 DQN 控制智能体：

$$a_t = \operatorname{argmax}_a Q(s_t, a; \mathbf{w}).$$

本章的 TD 算法用任意的行为策略收集 (s_t, a_t, r_t, s_{t+1}) 这样的四元组，然后拿它们训练目标策略，即 DQN。

行为策略和目标策略可以相同，也可以不同。同策略是指用相同的行为策略和目标策略；我们暂时还没有学到同策略。异策略是指用不同的行为策略和目标策略；本章的 DQN 是异策略。同策略和异策略如图 4.6、4.7 所示。

由于 DQN 是异策略，行为策略可以不同于目标策略，用任意的行为策略都可以，比如最常用的 ϵ -greedy：

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

让行为策略带有随机性的好处在于能探索更多没见过的状态。

异策略的好处是可以用行为策略收集经验，把 (s_t, a_t, r_t, s_{t+1}) 这样的四元组记录到一个数组里，在事后反复利用这些经验去更新目标策略。这个数组被称作经验回放数组 (Replay Buffer)，这种训练方式被称作经验回放 (Experience Replay)。注意，经验回放只适用于异策略，不适用于同策略，其原因是收集经验时用的行为策略不同于事后训练的目标策略。



图 4.6: 同策略。

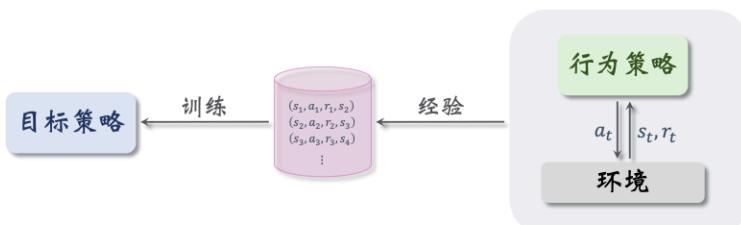


图 4.7: 异策略。

相关文献

DQN 首先由 Mnih 等人在 2013 年提出 [33]，其训练用的算法与本章介绍的基本一致，这种简单的训练算法实践中效果不佳。这篇论文用 Atari 游戏评价 DQN 的表现，虽然 DQN 的表现优于已有方法，但是它还是比人类的表现差一截。同样的作者在 2015 年发表了 DQN 的改进版本 [34]，其主要改进在于使用“目标网络”(Target Network)；这个版本的 DQN 在 Atari 游戏上的表现超越了人类玩家。

DQN 的本质是对最优动作价值函数 Q_* 的函数近似。早在 1995 年和 1997 年发表的论文 [3, 52] 就把函数近似用于价值学习中。本章使用的 TD 算法叫做 Q 学习算法，它是由 Watkins 在 1989 年在博士论文 [59] 提出的。Watkins 和 Dayan 发表在 1992 年的论文 [58] 分析了 Q 学习的收敛。1994 年的论文 [26, 51] 改进了 Q 学习算法的收敛分析。训练 DQN 用到的经验回放是由 Lin 在 1993 年的博士论文 [30] 中提出的。

参考文献

- [1] M. S. Abdulla and S. Bhatnagar. Reinforcement learning based algorithms for average cost markov decision processes. *Discrete Event Dynamic Systems*, 17(1):23–52, 2007.
- [2] L. V. Allis et al. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen Wageningen, 1994.
- [3] L. Baird. Residual algorithms: reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.
- [4] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.
- [5] P. Baudiš and J.-l. Gailly. Pachi: State of the art open source go program. In *Advances in computer games*, pages 24–38. Springer, 2011.
- [6] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.
- [7] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.
- [8] S. Bhatnagar and S. Kumar. A simultaneous perturbation stochastic approximation-based actor-critic algorithm for markov decision processes. *IEEE Transactions on Automatic Control*, 49(4):592–598, 2004.
- [9] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.
- [10] B. Bouzy and B. Helmstetter. Monte-Carlo go developments. In *Advances in computer games*, pages 159–174. Springer, 2004.
- [11] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfschagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.
- [13] M. Buro. From simple features to sophisticated evaluation functions. In *International Conference on Computers and Games*, pages 126–145. Springer, 1998.
- [14] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
- [15] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-Carlo tree search: A new framework for game AI. In *AIIDE*, 2008.
- [16] G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, and H. J. Van Den Herik. Monte-Carlo strategies for computer Go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, 2006.
- [17] G. M. J.-B. C. Chaslot. *Monte-Carlo tree search*. Maastricht University, 2010.
- [18] A. R. Conn, N. I. Gould, and P. L. Toint. *Trust region methods*. SIAM, 2000.
- [19] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.
- [20] R. Coulom. Computing “elo ratings” of move patterns in the game of Go. *ICGA journal*, 30(4):198–208, 2007.
- [21] M. Enzenberger, M. Müller, B. Arneson, and R. Segal. Fuego: an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.
- [22] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, et al. Noisy networks for exploration. In *International Conference on Learning Representations (ICLR)*, 2018.
- [23] R. Hafner and M. Riedmiller. Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169, 2011.
- [24] M. Hausknecht and P. Stone. Deep recurrent Q-learning for partially observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 2015.

- [25] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [26] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201, 1994.
- [27] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.
- [28] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.
- [29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.
- [30] L.-J. Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.
- [31] P. Marbach and J. N. Tsitsiklis. Simulation-based optimization of markov reward processes: Implementation issues. In *Proceedings of the 38th IEEE Conference on Decision and Control*, 1999.
- [32] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.
- [33] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [35] M. Müller. Computer go. *Artificial Intelligence*, 134(1-2):145–179, 2002.
- [36] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- [37] D. V. Prokhorov and D. C. Wunsch. Adaptive critic designs. *IEEE transactions on Neural Networks*, 8(5):997–1007, 1997.
- [38] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.
- [39] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.
- [40] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.
- [41] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2-3):273–289, 1992.
- [42] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2015.
- [43] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.
- [44] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [45] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, 2014.
- [46] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.
- [47] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In

- Advances in Neural Information Processing Systems (NIPS)*, 1996.
- [48] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
 - [49] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 2000.
 - [50] G. Tesauro and G. R. Galperin. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems*, pages 1068–1074, 1997.
 - [51] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine learning*, 16(3):185–202, 1994.
 - [52] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.
 - [53] H. J. Van Den Herik, J. W. Uiterwijk, and J. Van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, 2002.
 - [54] H. van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.
 - [55] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.
 - [56] H. van Seijen. Effective multi-step temporal-difference learning for non-linear function approximation. *arXiv preprint arXiv:1608.05151*, 2016.
 - [57] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.
 - [58] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
 - [59] C. J. C. H. Watkins. Learning from delayed rewards. 1989.
 - [60] R. J. Williams. *Reinforcement-learning connectionist systems*. College of Computer Science, Northeastern University, 1987.
 - [61] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
 - [62] Z. Yang, Y. Chen, M. Hong, and Z. Wang. Provably global convergence of actor-critic: A case for linear quadratic regulator with ergodic cost. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8353–8365, 2019.