

第十章 连续控制

本书前面章节的内容全部都是离散控制，即动作空间是一个离散的集合，比如超级玛丽游戏中的动作空间 $\mathcal{A} = \{\text{左, 右, 上}\}$ 就是个离散集合。本章的内容是连续控制，即动作空间是个连续集合，比如汽车的转向 $\mathcal{A} = [-90^\circ, 90^\circ]$ 就是连续集合。如果把连续动作空间做离散化，那么离散控制的方法就能直接解决连续控制问题；我们在第10.1节讨论连续集合的离散化。然而更好的办法是直接用连续控制方法，而非离散化之后借用离散控制方法。本章介绍两种连续控制方法：第10.2节介绍确定策略网络，第10.4节介绍随机策略网络。

10.1 离散控制与连续控制的区别

考虑这样一个问题：我们需要控制一只机械手臂，完成某些任务，获取奖励。机械手臂有两个关节，分别可以在 $[0^\circ, 360^\circ]$ 与 $[0^\circ, 180^\circ]$ 的范围内转动。这个问题的自由度是 $d = 2$ ，动作是二维向量，动作空间是连续集合 $\mathcal{A} = [0, 360] \times [0, 180]$ 。

此前我们学过的强化学习方法全部都是针对离散动作空间，不能直接解决上述连续控制问题。想把此前学过的离散控制方法应用到连续控制上，必须要对连续动作空间做离散化（网格化）。比如把连续集合 $\mathcal{A} = [0, 360] \times [0, 180]$ 变成离散集合 $\mathcal{A}' = \{0, 20, 40, \dots, 360\} \times \{0, 20, 40, \dots, 180\}$ ；见图 10.1。

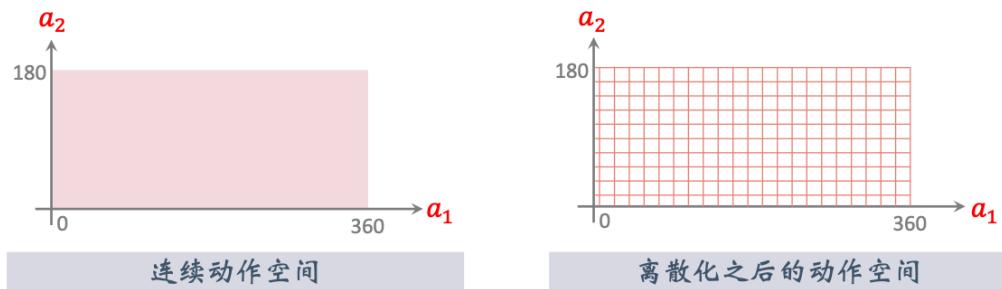


图 10.1：对连续动作空间 $\mathcal{A} = [0, 360] \times [0, 180]$ 做离散化（网格化）。

对动作空间做离散化之后，就可以应用之前学过的方法训练 DQN 或者策略网络，用于控制机械手臂。可是用离散化解决连续控制问题有个缺点。把自由度记作 d 。自由度 d 越大，网格上的点就越多，而且数量随着 d 指数增长，会造成维度灾难。当 d 等于几十的时候，网格上的点就是天文数字了。由于动作空间太大，DQN 和策略网络的训练都变得很困难，强化学习的结果肯定不好。上述离散化方法只适用于自由度 d 很小的情况下；如果 d 很大，就应该使用连续控制方法。后面两节介绍两种连续控制的方法。

10.2 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。DPG 是一种 Actor-Critic 方法，它有一个策略网络（演员），一个价值网络（评委）。策略网络控制智能体做运动，它基于状态 s 做出动作 \mathbf{a} 。¹ 价值网络不控制智能体，只是基于状态 s 给动作 \mathbf{a} 打分，从而指导策略网络做出改进。图 10.2 是两个神经网络的关系。

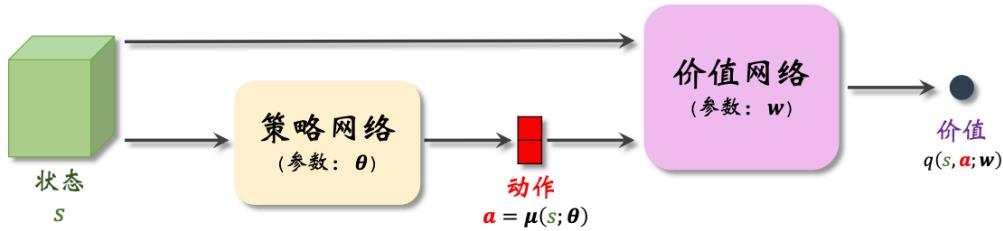


图 10.2：确定策略梯度 (DPG) 方法的示意图。策略网络 $\mu(s; \theta)$ 的输入是状态 s ，输出是动作 \mathbf{a} (d 维向量)。价值网络 $q(s, \mathbf{a}; \mathbf{w})$ 的输入是状态 s 和动作 \mathbf{a} ，输出是价值 (实数)。

10.2.1 策略网络和价值网络

本节的策略网络完全不同于前面章节的策略网络。在之前章节里，策略网络 $\pi(a|s; \theta)$ 是一个概率密度函数，它输出的是概率值。本节的确定策略网络 $\mu(s; \theta)$ 的输出是 d 维的向量 \mathbf{a} ，作为动作。两种策略网络一个是随机的，一个是确定性的：

- 之前章节中的策略网络 $\pi(a|s; \theta)$ 带有随机性：给定状态 s ，策略网络输出的是离散动作空间 \mathcal{A} 上的概率分布； \mathcal{A} 中的每个元素（动作）都有一个概率值。智能体依据概率分布，随机从 \mathcal{A} 中抽取一个动作，并执行动作。
- 本节的确定策略网络没有随机性：对于确定的状态 s ，策略网络 μ 输出的动作 \mathbf{a} 是确定的。动作 \mathbf{a} 直接是 μ 的输出，而非随机抽样得到的。

确定策略网络 μ 的结构如图 10.3 所示。如果输入的状态 s 是个矩阵或者张量（例如图片、视频），那么 μ 就是个普通的卷积神经网络，由若干卷积层、全连接层等组成。

本节的确定策略可以看做是随机策略的一个特例。确定策略 $\mu(s; \theta)$ 的输出是 d 维向量，它的第 i 个元素记作 $\hat{\mu}_i = [\mu(s; \theta)]_i$ 。定义下面这个随机策略：

$$\pi(\mathbf{a}|s; \theta, \sigma) = \prod_{i=1}^d \frac{1}{\sqrt{6.28\sigma_i}} \cdot \exp\left(-\frac{[a_i - \hat{\mu}_i]^2}{2\sigma_i^2}\right). \quad (10.1)$$

这个随机策略是均值为 $\mu(s; \theta)$ 、协方差矩阵为 $\text{diag}(\sigma_1, \dots, \sigma_d)$ 的多元正态分布。本节的确定策略可以看做是上述随机策略在 $\sigma = [\sigma_1, \dots, \sigma_d]$ 为全零向量时的特例。

本节的价值网络 $q(s, \mathbf{a}; \mathbf{w})$ 是对动作价值函数 $Q_\pi(s, \mathbf{a})$ 的近似。价值网络的结构如图 10.4 所示。价值网络的输入是状态 s 和动作 \mathbf{a} ，输出的价值 $\hat{q} = q(s, \mathbf{a}; \mathbf{w})$ 是个实数，可以反映动作的好坏；动作 \mathbf{a} 越好，则价值 \hat{q} 就越大。所以价值网络可以评价策略网络的表现。

¹本节中，动作 \mathbf{a} 是一个 d 维向量，而不是离散集合中的一个元素。因此本节用粗体 \mathbf{a} 表示动作的观测值。

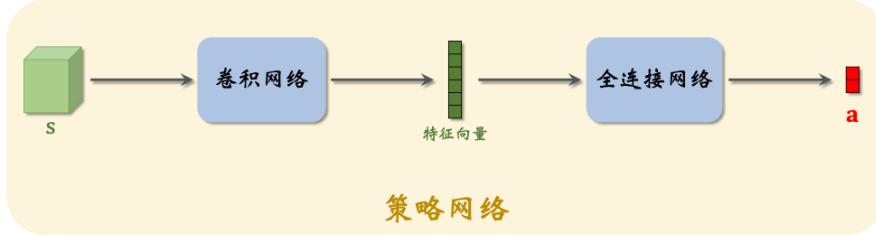


图 10.3: 确定策略网络 $\mu(s; \theta)$ 的结构。输入是状态 s , 输出是动作 a 。

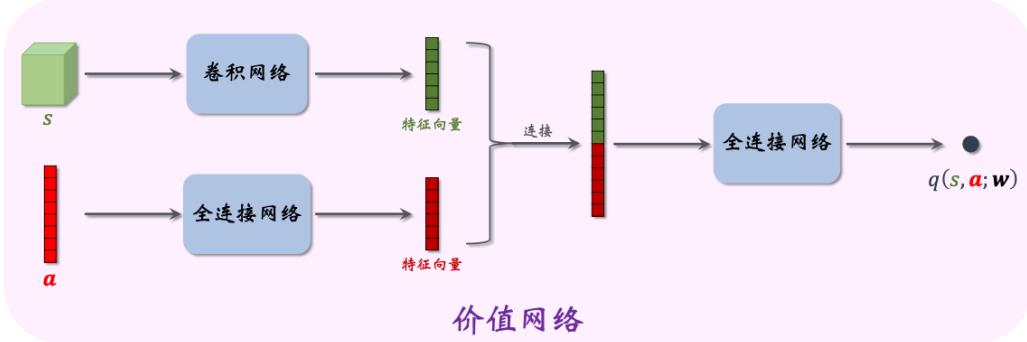


图 10.4: 价值网络 $q(s, a; w)$ 的结构。输入是状态 s 和动作 a , 输出是实数。

在训练的过程中，价值网络帮助训练策略网络。在训练结束之后，价值网络就被丢弃，由策略网络对智能体做控制。

10.2.2 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 (Behavior Policy) 可以不同于目标策略 (Target Policy)。目标策略即确定策略网络 $\mu(s; \theta)$ ，其中 θ 是策略网络最新的参数。行为策略可以是任意的，比如

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

公式的意思是行为策略可以用过时的策略网络参数，而且可以往动作中加入噪声 $\epsilon \in \mathbb{R}^d$ 。异策略的好处在于可以把收集经验与训练神经网络分割开；把收集到的经验存入经验回放数组 (Replay Buffer)，在做训练的时候重复利用收集到的经验。

用行为策略控制智能体与环境交互，把智能体的轨迹 (Trajectory) 整理成 (s_t, a_t, r_t, s_{t+1}) 这样的四元组，存入经验回放数组。在训练的时候，随机从数组中抽取一个四元组，记作 (s_j, a_j, r_j, s_{j+1}) 。在训练策略网络 $\mu(s; \theta)$ 的时候，只用到状态 s_j 。在训练价值网络 $q(s, a; w)$ 的时候，要用到四元组中全部四个元素： s_j, a_j, r_j, s_{j+1} 。

训练策略网络：首先通俗解释训练策略网络的原理。如图 10.5 所示，给定状态 s ，策略网络输出一个动作 $a = \mu(s; \theta)$ ，然后价值网络会给 a 打一个分数： $\hat{q} = q(s, a; w)$ 。参数 θ 影响 a ，从而影响 \hat{q} 。给定状态 s ，分数 \hat{q} 可以反映出 θ 的好坏程度。训练策略网络的目标就是改进参数 θ ，使得 \hat{q} 更高。把策略网络看做演员，价值网络看做评委。训

练习演员（策略网络）的目的就是让他迎合评委（价值网络）的喜好，改变自己的表演技巧（即参数 θ ），使得评委打分 \hat{q} 的均值更高。

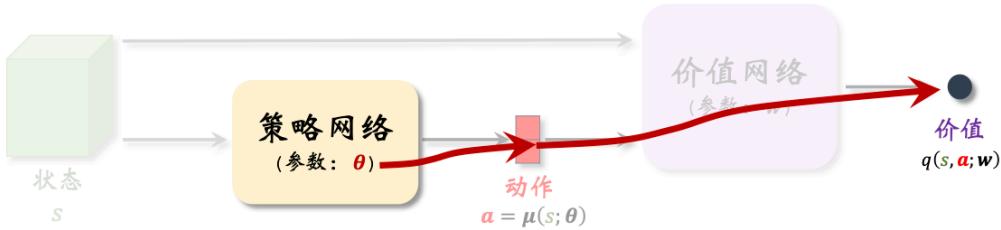


图 10.5: 给定状态 s , 策略网络的参数 θ 会影响 a , 从而影响 $\hat{q} = q(s, a; w)$ 。

根据以上解释, 我们来推导目标函数。如果当前状态是 s , 那么价值网络的打分就是:

$$q(s, \mu(s; \theta); w).$$

我们希望打分的期望尽量高, 所以把目标函数定义为打分的期望:

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

关于状态 S 求期望消除掉了 S 的影响; 不管面对什么样的状态 S , 演员 (策略网络) 都应该做出很好的动作, 使得平均分 $J(\theta)$ 尽量高。策略网络的学习可以建模成这样一个最大化问题:

$$\max_{\theta} J(\theta).$$

注意, 这里我们只训练策略网络, 所以最大化问题中的优化变量是策略网络的参数 θ , 而价值网络的参数 w 被固定住。

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:

$$\mathbf{g}_j \triangleq \nabla_{\theta} q(s_j, \mu(s_j; \theta); w).$$

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

可以用链式法则求出梯度 \mathbf{g}_j 。复习一下链式法则。如果有这样的函数关系: $\theta \rightarrow a \rightarrow q$, 那么 q 关于 θ 的导数可以写成

$$\frac{\partial q}{\partial \theta} = \frac{\partial a}{\partial \theta} \cdot \frac{\partial q}{\partial a}$$

由此我们得到下面的定理。

定理 10.1. 确定策略梯度

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



由此我们得到更新 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 $\hat{\mathbf{a}}_j = \mu(s_j; \boldsymbol{\theta})$ 。用梯度上升更新一次 $\boldsymbol{\theta}$:

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

此处的 β 是学习率，需要手动调。这样做随机梯度上升，可以逐渐让目标函数 $J(\boldsymbol{\theta})$ 增大，也就是让评委给演员的平均打分更高。

训练价值网络：首先通俗解释训练价值网络的原理。训练价值网络的目标是让价值网络 $q(s, \mathbf{a}; \mathbf{w})$ 的预测越来越接近真实价值函数 $Q_{\pi}(s, \mathbf{a})$ 。如果把价值网络看做评委，那么训练评委的目标就是让他的打分越来越准确。每一轮训练都要用到一个实际观测的奖励 r ，可以把 r 看做“真理”，用它来校准评委的打分。

训练价值网络要用 TD 算法。这里的 TD 算法与之前学过的标准 Actor-Critic 类似，都是让价值网络去拟合 TD 目标。每次从经验回放数组中取出一个四元组 $(s_j, \mathbf{a}_j, r_j, s_{j+1})$ ，用它更新一次参数 \mathbf{w} 。首先让价值网络做预测：

$$\hat{q}_j = q(s_j, \mathbf{a}_j; \mathbf{w}) \quad \text{和} \quad \hat{q}_{j+1} = q(s_{j+1}, \mu(s_{j+1}; \boldsymbol{\theta}); \mathbf{w}).$$

计算 TD 目标 $\hat{y}_j = r_j + \gamma \cdot \hat{q}_{j+1}$ 。把 $L(\mathbf{w}) = \frac{1}{2} [q(s_j, \mathbf{a}_j; \mathbf{w}) - \hat{y}_j]^2$ 作为损失函数，计算梯度

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = (\hat{q}_j - \hat{y}_j) \cdot \nabla_{\mathbf{w}} q(s_j, \mathbf{a}_j; \mathbf{w}),$$

做一轮梯度下降更新参数 \mathbf{w} :

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

这样可以让损失函数 $L(\mathbf{w})$ 减小，也就是让价值网络的预测 \hat{q}_j 更接近 TD 目标 \hat{y}_j 。公式中的 α 是学习率，需要手动调。

训练流程：做训练的时候，可以同时对价值网络和策略网络做训练。每次从经验回放数组中抽取一个四元组，记作 $(s_j, \mathbf{a}_j, r_j, s_{j+1})$ 。把神经网络当前参数记作 \mathbf{w}_{now} 和 $\boldsymbol{\theta}_{\text{now}}$ 。执行以下步骤更新策略网络和价值网络：

1. 让策略网络做预测：

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

注 计算动作 $\hat{\mathbf{a}}_j$ 用的是当前的策略网络 $\mu(s_j; \boldsymbol{\theta}_{\text{now}})$ ，用 $\hat{\mathbf{a}}_j$ 来更新 $\boldsymbol{\theta}_{\text{now}}$ ；而从经验回放数组中抽取的 \mathbf{a}_j 则是用过时的策略网络 $\mu(s_j; \boldsymbol{\theta}_{\text{old}})$ 算出的，用 \mathbf{a}_j 来更新 \mathbf{w}_{now} 。请注意 $\hat{\mathbf{a}}_j$ 与 \mathbf{a}_j 的区别。

2. 让价值网络做预测：

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

3. 计算 TD 目标和 TD 误差：

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

4. 更新价值网络：

$$\mathbf{w}_{\text{new}} \leftarrow \mathbf{w}_{\text{now}} - \alpha \cdot \delta_j \cdot \nabla_{\mathbf{w}} q(s_j, \mathbf{a}_j; \mathbf{w}_{\text{now}}).$$

5. 更新策略网络：

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{now}} + \beta \cdot \nabla_{\boldsymbol{\theta}} \mu(s_j; \boldsymbol{\theta}_{\text{now}}) \cdot \nabla_{\boldsymbol{a}} q(s_j, \hat{\boldsymbol{a}}_j; \boldsymbol{w}_{\text{now}}).$$

在实践中，上述算法的表现并不好；读者应当采用下一节介绍的技巧训练策略网络和价值网络。

10.3 双延时确定策略梯度 (TD3)

上一节介绍的算法实际效果并不好。本节介绍的 Twin Delayed Deep Deterministic Policy Gradient (TD3) 可以大幅提升算法的表现，把策略网络和价值网络训练得更好。注意，本节只是改进训练用的算法，并不改变神经网络的结构。

10.3.1 高估问题及其原因

第 6.2 节讨论过用 Q 学习算法训练 DQN 会导致高估。上一节的训练算法也出现相同的高估问题，下面我们分析造成高估的原因。训练策略网络的时候，我们希望策略网络计算出的动作 $\hat{\mathbf{a}} = \mu(s; \boldsymbol{\theta})$ 能得到价值网络尽量高的评价，也就是让 $q(s, \hat{\mathbf{a}}; \mathbf{w})$ 尽量大。我们通过求解下面的优化模型来学习策略网络：

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathbb{E}_S \left[q(S, \hat{A}; \mathbf{w}) \right], \quad \text{s.t. } \hat{A} = \mu(S; \boldsymbol{\theta}).$$

这个公式的意思是 $\mu(s; \boldsymbol{\theta}^*)$ 是最优的确定策略。上面的公式与下面的公式意义相同（虽然不严格等价）：

$$\mu(s; \boldsymbol{\theta}^*) = \underset{\mathbf{a} \in \mathcal{A}}{\operatorname{argmax}} q(s, \mathbf{a}; \mathbf{w}), \quad \forall s \in \mathcal{S}.$$

这个公式的意思也是 $\mu(s; \boldsymbol{\theta}^*)$ 是最优的确定策略。

最大化造成高估： 回忆一下，DQN 的决策方式是 $a_j = \operatorname{argmax}_a Q(s_j, a; \mathbf{w})$ 。上面的分析说明确定策略 μ 做决策的方式是 $\mathbf{a}_j \approx \operatorname{argmax}_{\mathbf{a}} q(s_j, \mathbf{a}; \mathbf{w})$ 。不难看出，DQN 与确定策略的决策方式都是选择最大化价值的动作。

第 6.2 节讨论过造成 DQN 的高估问题的原因：最大化会造成 TD 目标 \hat{y}_j 高估真实最优动作价值 $Q_*(s_j, a_j)$ 。同理，训练价值网络 q 用的 TD 目标

$$\begin{aligned} \hat{y}_j &= r_j + \gamma \cdot q(s_{j+1}, \mu(s_{j+1}; \boldsymbol{\theta}); \mathbf{w}) \\ &\approx r_j + \gamma \cdot \max_{\mathbf{a}_{j+1}} q(s_{j+1}, \mathbf{a}_{j+1}; \mathbf{w}) \end{aligned}$$

是对真实动作价值的高估。TD 算法把 \hat{y}_j 作为目标，鼓励价值网络 $q(s_j, a_j; \mathbf{w})$ 接近 \hat{y}_j 。这回导致 $q(s_j, a_j; \mathbf{w})$ 高估真实动作价值。

自举造成偏差传导： 我们在第 6.2 节讨论过自举 (Bootstrapping) 造成偏差的传导。TD 目标

$$\hat{y}_j = r_j + \gamma \cdot q(s_{j+1}, \mu(s_{j+1}; \boldsymbol{\theta}); \mathbf{w})$$

是用价值网络算出来的，而它又被用于更新价值网络 q 本身，这属于自举。假如价值网络 $q(s_{j+1}, \mathbf{a}_{j+1}; \boldsymbol{\theta})$ 高估了真实动作价值 $Q_\pi(s_{j+1}, \mathbf{a}_{j+1})$ ，那么 TD 目标 \hat{y}_j 则是对 $Q_\pi(s_j, \mathbf{a}_j)$ 的高估，这会导致 $q(s_j, a_j; \mathbf{w})$ 高估 $Q_\pi(s_j, \mathbf{a}_j)$ 。自举让高估从 $(s_{j+1}, \mathbf{a}_{j+1})$ 传播到 (s_j, \mathbf{a}_j) 。

10.3.2 高估问题的解决方案

解决方案——目标网络：为了解决自举和最大化造成的问题，我们需要使用目标网络 (Target Networks) 计算 TD 目标 \hat{y}_j 。训练中需要两个目标网络：

$$q(s, \mathbf{a}; \mathbf{w}^-) \text{ 和 } \mu(s; \boldsymbol{\theta}^-).$$

它们与价值网络、策略网络的结构完全相同，但是参数不同。TD 目标是用目标网络算的：

$$\hat{y}_j = r_j + \gamma \cdot q(s_{j+1}, \hat{\mathbf{a}}_{j+1}; \mathbf{w}^-), \quad \text{其中 } \hat{\mathbf{a}}_{j+1} = \mu(s_{j+1}; \boldsymbol{\theta}^-).$$

把 \hat{y}_j 作为目标，更新 \mathbf{w} ，鼓励 $q(s_j, a_j; \mathbf{w})$ 接近 \hat{y}_j 。四个神经网络之间的关系如图 10.6 所示。这种方法可以在一定程度上缓解高估，但是实验表明高估仍然很严重。

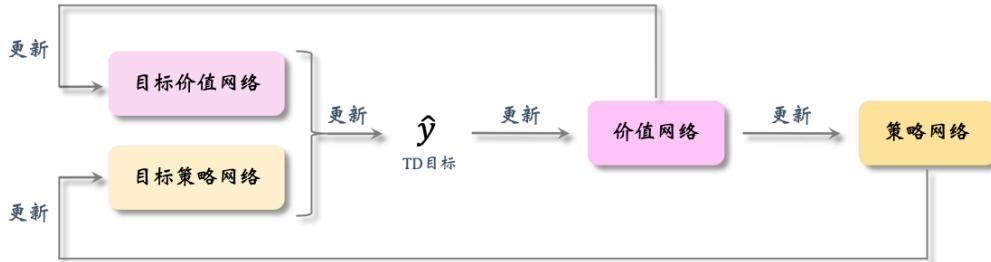


图 10.6: 四个神经网络之间的关系。

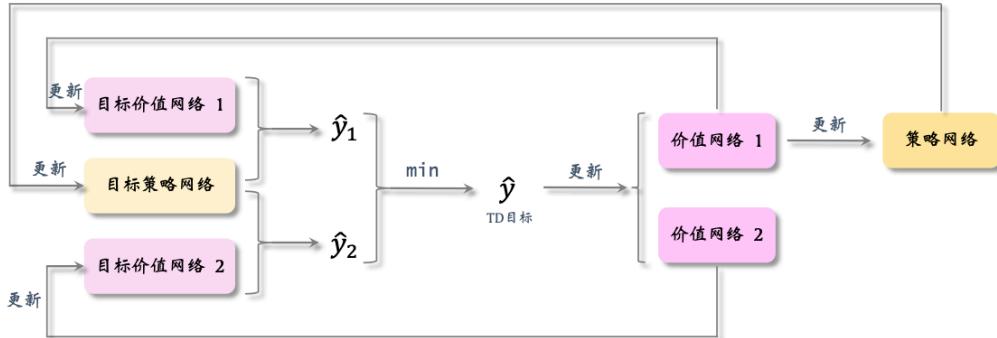


图 10.7: 截断双 Q 学习算法中六个神经网络之间的关系。

更好的解决方案——截断双 Q 学习：这种方法使用两个价值网络和一个策略网络：

$$q(s, \mathbf{a}; \mathbf{w}_1), \quad q(s, \mathbf{a}; \mathbf{w}_2), \quad \mu(s; \boldsymbol{\theta}).$$

三个神经网络各对应一个目标网络：

$$q(s, \mathbf{a}; \mathbf{w}_1^-), \quad q(s, \mathbf{a}; \mathbf{w}_2^-), \quad \mu(s; \boldsymbol{\theta}^-).$$

用目标策略网络计算动作：

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

然后用两个目标价值网络计算：

$$\begin{aligned}\hat{y}_{j,1} &= r_j + \gamma \cdot q(s_{j+1}, \hat{\mathbf{a}}_{j+1}^-; \mathbf{w}_1^-), \\ \hat{y}_{j,2} &= r_j + \gamma \cdot q(s_{j+1}, \hat{\mathbf{a}}_{j+1}^-; \mathbf{w}_2^-).\end{aligned}$$

取两者较小者为 TD 目标：

$$\hat{y}_j = \min \left\{ \hat{y}_{j,1}, \hat{y}_{j,2} \right\}.$$

截断双 Q 学习中的六个神经网络的关系如图 10.7 所示。

10.3.3 其他改进方法

可以在截断双 Q 学习算法的基础上做两处小的改进，进一步提升算法的表现。两种改进分别是往动作中加噪声、减小更新策略网络和目标网络的频率。

往动作中加噪声：上一小节中截断双 Q 学习用目标策略网络计算动作： $\hat{\mathbf{a}}_{j+1}^- = \mu(s_{j+1}; \theta^-)$ 。把这一步改成：

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

公式中的 ξ 是个随机向量，表示噪声，它的每一个元素独立随机从截断正态分布 (Clipped Normal Distribution) 中抽取。把截断正态分布记作 $\mathcal{CN}(0, \sigma^2, -c, c)$ ，意思是均值为零，标准差为 σ 的正态分布，但是变量落在区间 $[-c, c]$ 之外的概率为零。正态分布与截断正态分布的对比如图 10.8 所示。使用截断正态分布，而非正态分布，是为了防止噪声 ξ 过大。使用截断，保证噪声大小不会超过 $-c$ 和 c 。

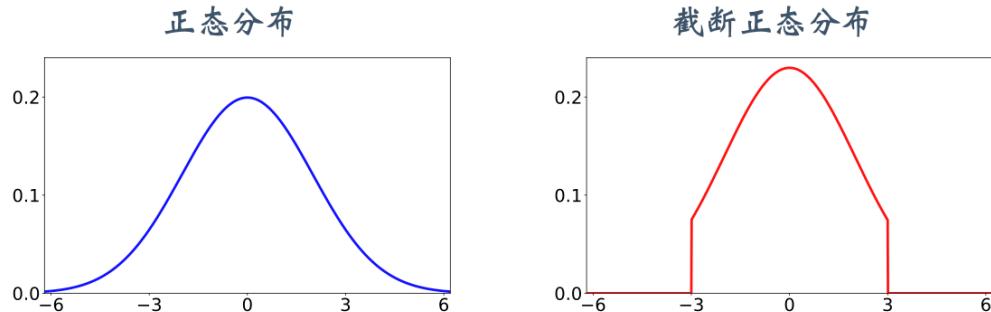


图 10.8：正态分布 $N(0, 1^2)$ 和截断正态分布 $\mathcal{CN}(0, 1^2, -3, 3)$ 。

减小更新策略网络和目标网络的频率：Actor-Critic 用价值网络来指导策略网络的更新。如果价值网络 q 本身不可靠，那么用价值网络 q 给动作打的分数是不准确的，无助于改进策略网络 μ 。在价值网络 q 还很差的时候就急于更新 μ ，非但不能改进 μ ，反而会由于 μ 的变化导致 q 的训练不稳定。

实验表明，应当让策略网络 μ 以及三个目标网络的更新慢于价值网络 q 。传统的 Actor-Critic 的每一轮训练都对策略网络、价值网络、以及目标网络做一次更新。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。 k 是超参数，需要调。

10.3.4 训练流程

本节介绍了三种技巧，改进 DPG 的训练。第一，用截断双 Q 学习，缓解价值网络的高估。第二，往目标策略网络中加噪声，起到平滑作用。第三，降低策略网络和三个目标网络的更新频率。使用这三种技巧的算法被称作双延时确定策略梯度 (Twin Delayed Deep Deterministic Policy Gradient)，缩写是 TD3。

TD3 与 DPG 都属于异策略 (Off-policy)，可以用任意的行为策略收集经验，事后做经验回放训练策略网络和价值网络。收集经验的方式与原始的训练算法相同，用 $a_t = \mu(s_t; \theta) + \epsilon$ 与环境交互，把观测到的四元组 (s_t, a_t, r_t, s_{t+1}) 存入经验回放数组。

初始的时候，策略网络和价值网络的参数都是随机的。这样初始化目标网络的参数：

$$\mathbf{w}_1^- \leftarrow \mathbf{w}_1, \quad \mathbf{w}_2^- \leftarrow \mathbf{w}_2, \quad \boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}.$$

训练策略网络和价值网络的时候，每次从数组中随机抽取一个四元组，记作 (s_j, a_j, r_j, s_{j+1}) 。然后执行下面的步骤，更新价值网络、策略网络、目标网络。

1. 让目标策略网络做预测： $\hat{a}_{j+1}^- = \mu(s_{j+1}; \boldsymbol{\theta}^-) + \xi$ ，其中 $\xi \sim \mathcal{CN}(0, \sigma^2, -c, c)$ 。

2. 让两个目标价值网络做预测

$$\hat{q}_{1,j+1}^- = q(s_{j+1}, \hat{a}_{j+1}^-; \mathbf{w}_{1,\text{now}}^-) \quad \text{和} \quad \hat{q}_{2,j+1}^- = q(s_{j+1}, \hat{a}_{j+1}^-; \mathbf{w}_{2,\text{now}}^-).$$

3. 计算 TD 目标：

$$\hat{y}_j = r_j + \gamma \cdot \min \left\{ \hat{q}_{1,j+1}^-, \hat{q}_{2,j+1}^- \right\}.$$

4. 让两个价值网络做预测：

$$\hat{q}_{1,j} = q(s_j, a_j; \mathbf{w}_{1,\text{now}}) \quad \text{和} \quad \hat{q}_{2,j} = q(s_j, a_j; \mathbf{w}_{2,\text{now}}).$$

5. 计算 TD 误差：

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

6. 更新价值网络：

$$\begin{aligned} \mathbf{w}_{1,\text{new}} &\leftarrow \mathbf{w}_{1,\text{now}} - \alpha \cdot \delta_{1,j} \cdot \nabla_{\mathbf{w}} q(s_j, a_j; \mathbf{w}_{1,\text{now}}), \\ \mathbf{w}_{2,\text{new}} &\leftarrow \mathbf{w}_{2,\text{now}} - \alpha \cdot \delta_{2,j} \cdot \nabla_{\mathbf{w}} q(s_j, a_j; \mathbf{w}_{2,\text{now}}). \end{aligned}$$

7. 每隔 k 轮更新一次策略网络和三个目标网络：

- 让策略网络做预测： $\hat{a}_j = \mu(s_j; \boldsymbol{\theta})$ 。然后更新策略网络：

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{now}} + \beta \cdot \nabla_{\boldsymbol{\theta}} \mu(s_j; \boldsymbol{\theta}_{\text{now}}) \cdot \nabla_{\boldsymbol{\theta}} q(s_j, \hat{a}_j; \mathbf{w}_{1,\text{now}}).$$

- 更新目标网络的参数：

$$\boldsymbol{\theta}_{\text{new}}^- \leftarrow \tau \boldsymbol{\theta}_{\text{new}} + (1 - \tau) \boldsymbol{\theta}_{\text{now}},$$

$$\mathbf{w}_{1,\text{new}}^- \leftarrow \tau \mathbf{w}_{1,\text{new}} + (1 - \tau) \mathbf{w}_{1,\text{now}}^-,$$

$$\mathbf{w}_{2,\text{new}}^- \leftarrow \tau \mathbf{w}_{2,\text{new}} + (1 - \tau) \mathbf{w}_{2,\text{now}}^-.$$

10.4 随机高斯策略

上一节用确定策略网络解决连续控制问题。本节用不同的方法做连续控制，本节的策略网络是随机的，它跟我们以前学过的策略网络区别非常大。

10.4.1 基本思路

我们先研究最简单的情形：自由度等于 1，也就是说动作 a 是实数，动作空间 $\mathcal{A} \subset \mathbb{R}$ 。把动作的均值记作 $\mu(s)$ ，标准差记作 $\sigma(s)$ ，它们都是状态 s 的函数。用正态分布的概率密度函数作为策略函数：

$$\pi(a|s) = \frac{1}{\sqrt{6.28 \cdot \sigma(s)}} \cdot \exp\left(-\frac{[a - \mu(s)]^2}{2 \cdot \sigma^2(s)}\right). \quad (10.2)$$

假如我们知道函数 $\mu(s)$ 和 $\sigma(s)$ 的解析表达式，可以这样做控制：

1. 观测到当前状态 s ，预测均值 $\hat{\mu} = \mu(s)$ 和标准差 $\hat{\sigma} = \sigma(s)$ 。
2. 从正态分布中做随机抽样： $a \sim \mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ ；智能体执行动作 a 。

然而我们并不知道 $\mu(s)$ 和 $\sigma(s)$ 是怎么样的函数。一个很自然的想法是用神经网络来近似这两个函数。把神经网络记作 $\mu(s; \theta)$ 和 $\rho(s; \theta)$ ，其中 θ 表示神经网络中的可训练参数。但实践中最好不要直接近似标准差 σ ，而是近似方差对数 $\ln \sigma^2$ 。定义两个神经网络：

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

分别用于预测均值和方差对数。可以按照图 10.10 来搭建神经网络。神经网络的输入是状态 s ，通常是向量、矩阵、或者张量。神经网络有两个输出头，分别记作 $\mu(s; \theta)$ 和 $\rho(s; \theta)$ 。可以这样用神经网络做控制：

1. 观测到当前状态 s ，计算均值 $\hat{\mu} = \mu(s; \theta)$ ，方差对数 $\hat{\rho} = \rho(s; \theta)$ ，以及方差 $\hat{\sigma}^2 = \exp(\hat{\rho})$ 。
2. 从正态分布中做随机抽样： $a \sim \mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ ；智能体执行动作 a 。

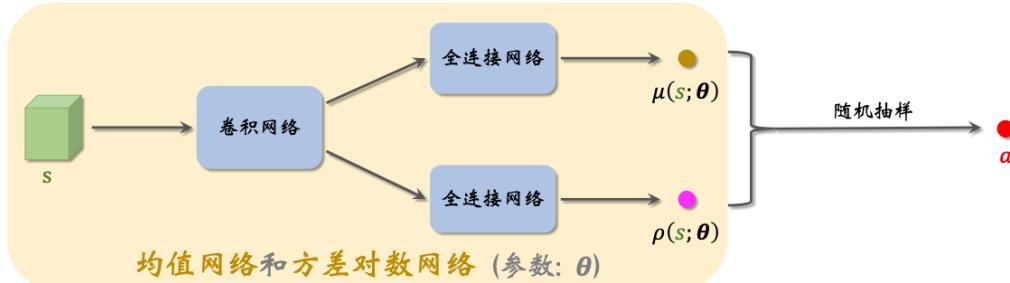


图 10.9：高斯策略网络有两个头，一个输出均值 $\hat{\mu}$ ，另一个输出方差对数 $\hat{\rho}$ 。

用神经网络近似均值和标准差之后，公式 (10.2) 中的策略函数 $\pi(a|s)$ 变成了下面的策略网络：

$$\pi(a|s; \theta) = \frac{1}{\sqrt{6.28 \cdot \exp[\rho(s; \theta)]}} \cdot \exp\left(-\frac{[a - \mu(s; \theta)]^2}{2 \cdot \exp[\rho(s; \theta)]}\right).$$

实际做控制的时候，我们只需要神经网络 $\mu(s; \theta)$ 和 $\rho(s; \theta)$ ，用不到真正的策略网络 $\pi(a|s; \theta)$ 。

10.4.2 随机高斯策略网络

上一小节假设控制问题的自由度是 $d = 1$ ，也就是说动作 a 都是标量。实际问题中的自由度 d 往往大于 1，那么动作 a 是 d 维向量。对于这样的问题，我们修改一下神经网络结构，让两个输出 $\mu(s; \theta)$ 和 $\rho(s; \theta)$ 都 d 维向量；见图10.10。

用标量 a_i 表示动作 a （向量）的第 i 个元素。用函数 $\mu_i(s; \theta)$ 和 $\rho_i(s; \theta)$ 分别表示 $\mu(s; \theta)$ 和 $\rho(s; \theta)$ 的第 i 个元素。我们用下面这个特殊的多元正态分布的概率密度函数表示作为策略网络：

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

做控制的时候只需要均值网络 $\mu(s; \theta)$ 和方差对数网络 $\rho(s; \theta)$ ，不需要策略网络 $\pi(a|s; \theta)$ 。做训练的时候也不需要 $\pi(a|s; \theta)$ ，而是要用辅助网络 $f(s, a; \theta)$ 。总而言之，策略网络 π 只是帮助你理解本节的方法而已，实际算法中不会出现 π 。

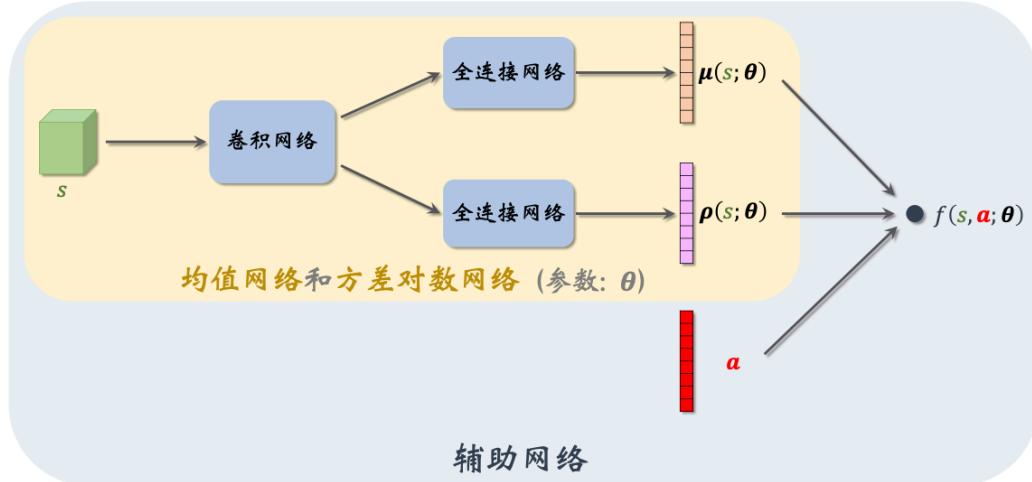


图 10.10：辅助网络的结构示意图。辅助神经网络的输入是状态 s 与动作 a ，输出是实数 $f(s, a; \theta)$ 。

图10.10描述了辅助网络 $f(s, a; \theta)$ 与 μ 、 ρ 、 a 的关系。辅助网络具体是这样定义的：

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

它的可训练参数 θ 都是从 $\mu(s; \theta)$ 和 $\rho(s; \theta)$ 中来的。不难发现，辅助网络与策略网络有这样的关系：

$$f(s, a; \theta) = \ln \pi(a|s; \theta) + \text{Constant.} \quad (10.3)$$

10.4.3 策略梯度

回忆一下之前学过的内容。在 t 时刻的折扣回报记作随机变量

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

动作价值函数 $Q_\pi(s_t, a_t)$ 是对折扣回报 U_t 的条件期望。前面章节推导过策略梯度的蒙特卡洛近似：

$$\mathbf{g} = Q_\pi(s, a) \cdot \nabla_{\boldsymbol{\theta}} \ln \pi(\mathbf{a}|s; \boldsymbol{\theta}).$$

由公式 (10.3) 可得：

$$\mathbf{g} = Q_\pi(s, a) \cdot \nabla_{\boldsymbol{\theta}} f(s, \mathbf{a}; \boldsymbol{\theta}). \quad (10.4)$$

有了策略梯度，就可以学习参数 $\boldsymbol{\theta}$ 。训练的过程大致如下：

1. 搭建均值网络 $\mu(s; \boldsymbol{\theta})$ 、方差对数网络 $\rho(s; \boldsymbol{\theta})$ 、辅助网络 $f(s, \mathbf{a}; \boldsymbol{\theta})$ 。
2. 让智能体与环境交互，记录每一步的状态、动作、奖励，并对参数 $\boldsymbol{\theta}$ 做更新：
 - (a). 观测到当前状态 s ，计算均值、方差对数、方差：

$$\hat{\mu} = \mu(s; \boldsymbol{\theta}), \quad \hat{\rho} = \rho(s; \boldsymbol{\theta}), \quad \hat{\sigma}^2 = \exp(\hat{\rho}).$$

此处的指数函数 $\exp(\cdot)$ 应用到向量的每一个元素上。

- (b). 设 $\hat{\mu}_i$ 和 $\hat{\sigma}_i^2$ 分别是 d 维向量 $\hat{\mu}$ 和 $\hat{\sigma}$ 的第 i 个元素。从正态分布中做抽样：

$$a_i \sim \mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i^2), \quad \forall i = 1, \dots, d.$$

把得到的动作记作 $\mathbf{a} = [a_1, \dots, a_d]$ 。

- (c). 计算动作价值： $q = Q_\pi(s, a)$ 。
- (d). 用反向传播计算出辅助网络关于参数 $\boldsymbol{\theta}$ 的梯度： $\nabla_{\boldsymbol{\theta}} f(s, \mathbf{a}; \boldsymbol{\theta})$ 。
- (e). 用策略梯度上升更新参数：

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \cdot q \cdot \nabla_{\boldsymbol{\theta}} f(s, \mathbf{a}; \boldsymbol{\theta}).$$

此处的 β 是学习率。

但是算法中有一个没解决的问题：我们并不知道动作价值 $Q_\pi(s, a)$ 。有两种办法近似 $Q_\pi(s, a)$ ：REINFORCE 用实际观测的折扣回报代替 $Q_\pi(s, a)$ ，Actor-Critic 用价值网络近似 Q_π 。后面两小节具体讲解这两种算法。

10.4.4 用 REINFORCE 学习参数 $\boldsymbol{\theta}$

REINFORCE 用实际观测的折扣回报 $u_t = \sum_{k=t}^n \gamma^{k-t} \cdot r_k$ 代替动作价值 $Q_\pi(s_t, \mathbf{a}_t)$ 。道理是这样的。动作价值是回报的期望：

$$Q_\pi(s_t, \mathbf{a}_t) = \mathbb{E}[U_t | S_t = s_t, A_t = \mathbf{a}_t].$$

随机变量 U_t 的一个实际观测值 u_t 是期望的蒙特卡洛近似。这样一来，公式 (10.4) 中的策略梯度就能近似成

$$\mathbf{g} \approx u_t \cdot \nabla_{\boldsymbol{\theta}} f(s, \mathbf{a}; \boldsymbol{\theta}).$$

在搭建好均值网络 $\mu(s; \boldsymbol{\theta})$ 、方差对数网络 $\rho(s; \boldsymbol{\theta})$ 、辅助网络 $f(s, \mathbf{a}; \boldsymbol{\theta})$ 之后，我们用

REINFORCE 更新参数 θ 。设当前参数为 θ_{now} 。REINFORCE 重复以下步骤，直到收敛：

- 用 $\mu(s; \theta_{\text{now}})$ 和 $\rho(s; \theta_{\text{now}})$ 控制智能体与环境交互，完成一局游戏，得到一条轨迹：

$$s_1, \mathbf{a}_1, r_1, s_2, \mathbf{a}_2, r_2, \dots, s_n, \mathbf{a}_n, r_n.$$

- 计算所有的回报：

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

- 对辅助网络做反向传播，得到所有的梯度：

$$\nabla_{\theta} f(s_t, \mathbf{a}_t; \theta_{\text{now}}), \quad \forall t = 1, \dots, n.$$

- 用策略梯度上升更新参数：

$$\theta_{\text{new}} \leftarrow \theta_{\text{now}} + \beta \cdot \sum_{t=1}^n \gamma^{t-1} \cdot u_t \cdot \nabla_{\theta} f(s_t, \mathbf{a}_t; \theta_{\text{now}})$$

上述算法标准的 REINFORCE，效果不如使用基线的 REINFORCE。读者可以参考第8.2节的内容，把状态价值作为基线，改进上面描述的算法。REINFORCE 算法属于同策略(On-policy)，不能使用经验回放。

10.4.5 用 Actor-Critic 学习参数 θ

Actor-Critic 需要搭建一个价值网络 $q(s, \mathbf{a}; \mathbf{w})$ ，用于近似动作价值函数 $Q_{\pi}(s, \mathbf{a})$ 。价值网络的结构如图 10.11 所示。此外，还需要一个目标价值网络 $q(s, \mathbf{a}; \mathbf{w}^-)$ ，网络结构相同，但是参数不同。

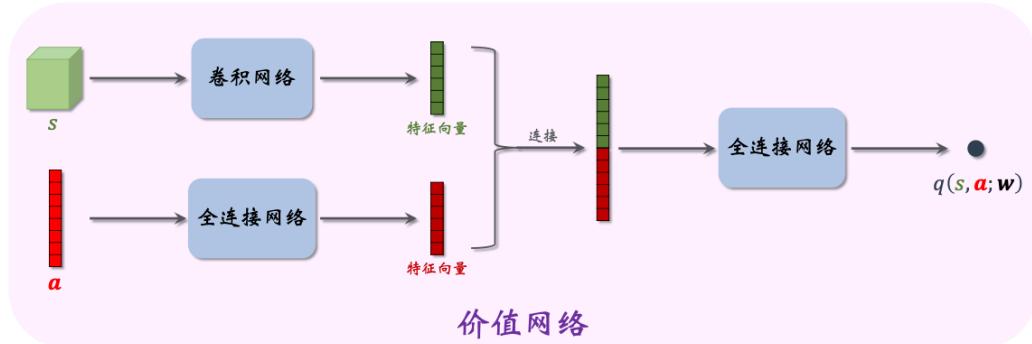


图 10.11：价值网络 $q(s, \mathbf{a}; \mathbf{w})$ 的结构。输入是状态 s 和动作 \mathbf{a} ，输出是实数。

在搭建好均值网络 μ 、方差对数网络 ρ 、辅助网络 f 、价值网络 q 之后，我们用 SARSA 算法更新价值网络参数 \mathbf{w} ，用近似策略梯度更新控制器参数 θ 。设当前参数为 \mathbf{w}_{now} 和 θ_{now} 。重复以下步骤更新价值网络的和控制器参数，直到收敛：

- 实际观测到当前状态 s_t ，用控制器算出均值 $\mu(s_t; \theta_{\text{now}})$ 和方差对数 $\rho(s_t; \theta_{\text{now}})$ ，然后随机抽样得到动作 \mathbf{a}_t 。智能体执行动作 \mathbf{a}_t ，观测到奖励 r_t 与新的状态 s_{t+1} 。
- 计算均值 $\mu(s_{t+1}; \theta_{\text{now}})$ 和方差对数 $\rho(s_{t+1}; \theta_{\text{now}})$ ，然后随机抽样得到动作 $\tilde{\mathbf{a}}_{t+1}$ 。这个动作只是假想动作，智能体不予执行。

3. 用价值网络计算出：

$$\hat{q}_t = q(s_t, \mathbf{a}_t; \mathbf{w}_{\text{now}}).$$

4. 用目标网络计算出：

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

5. 计算 TD 目标和 TD 误差：

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

6. 更新价值网络的参数：

$$\mathbf{w}_{\text{new}} \leftarrow \mathbf{w}_{\text{now}} - \alpha \cdot \delta_t \cdot \nabla_{\mathbf{w}} q(s_t, \mathbf{a}_t; \mathbf{w}_{\text{now}}).$$

7. 更新策略网络参数：

$$\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{now}} + \beta \cdot \hat{q}_t \cdot \nabla_{\boldsymbol{\theta}} f(s_t, \mathbf{a}_t; \boldsymbol{\theta}_{\text{now}})$$

8. 更新目标网络参数：

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

算法中的 α 、 β 、 τ 都是超参数，需要手动调整。上述算法标准的 Actor-Critic，效果不如 Advantage Actor-Critic (A2C)。读者可以参考第8.3节的内容，用 A2C 改进上面描述的算法。

相关文献

确定策略梯度 (Deterministic Policy Gradient, DPG) 方法由 David Silver 等学者在 2014 年的文章中提出 [57]。随后同一批作者把相似的想法与深度学习结合起来，方法叫作深度确定策略梯度 (Deep Deterministic Policy Gradient, DPG)，文章在 2016 年发表 [38]。这两篇论文使得 DPG 方法流行起来。但值得注意的是，相似的想法在更早的论文中有提出：[29, 48]。

2018 年的论文 [27] 提出三种对 DPG 的改进方法，并将改进的算法命名为 TD3。2017 年的论文 [28] 提出了 Soft Actor-Critic (SAC)，也可以解决连续控制问题。

Degriz 等人在 2012 年发表的论文 [23] 使用正态分布的概率密度函数作为策略函数，并且用线性函数近似均值和方差对数。类似的连续控制方法最早由 Williams 在 1987 和 1992 年提出 [73-74]。

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] Z. Ahmed, N. Le Roux, M. Norouzi, and D. Schuurmans. Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning (ICML)*, 2019.
- [3] L. V. Allis et al. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen Wageningen, 1994.
- [4] L. Baird. Residual algorithms: reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.
- [5] 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.
- [6] 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.
- [7] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.
- [8] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.
- [9] 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.
- [10] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.
- [11] B. Bouzy and B. Helmstetter. Monte-Carlo go developments. In *Advances in computer games*, pages 159–174. Springer, 2004.
- [12] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [13] 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.
- [14] M. Buro. From simple features to sophisticated evaluation functions. In *International Conference on Computers and Games*, pages 126–145. Springer, 1998.
- [15] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
- [16] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-Carlo tree search: A new framework for game AI. In *AIIDE*, 2008.
- [17] 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.
- [18] G. M. J.-B. C. Chaslot. *Monte-Carlo tree search*. Maastricht University, 2010.
- [19] Y. Chow, O. Nachum, and M. Ghavamzadeh. Path consistency learning in Tsallis entropy regularized mdps. In *International Conference on Machine Learning (ICML)*, pages 979–988, 2018.
- [20] A. R. Conn, N. I. Gould, and P. L. Toint. *Trust region methods*. SIAM, 2000.
- [21] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.
- [22] R. Coulom. Computing “elo ratings” of move patterns in the game of Go. *ICGA journal*, 30(4):198–208, 2007.
- [23] T. Degris, P. M. Pilarski, and R. S. Sutton. Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC)*, 2012.
- [24] 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.

- [25] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018.
- [26] 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.
- [27] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, 2018.
- [28] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*, 2017.
- [29] R. Hafner and M. Riedmiller. Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169, 2011.
- [30] 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.
- [31] 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.
- [32] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [33] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.
- [34] 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.
- [35] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.
- [36] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.
- [37] K. Lee, S. Choi, and S. Oh. Sparse Markov decision processes with causal sparse Tsallis entropy regularization for reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):1466–1473, 2018.
- [38] 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.
- [39] L.-J. Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.
- [40] P. Marbach and J. N. Tsitsiklis. Simulation-based optimization of Markov reward processes: Implementation issues. In *IEEE Conference on Decision and Control*, 1999.
- [41] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.
- [42] 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.
- [43] 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.
- [44] 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.
- [45] M. Müller. Computer go. *Artificial Intelligence*, 134(1-2):145–179, 2002.
- [46] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- [47] B. O’Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih. Combining policy gradient and Q-learning. In *International Conference on Learning Representations (ICLR)*, 2017.
- [48] D. V. Prokhorov and D. C. Wunsch. Adaptive critic designs. *IEEE transactions on Neural Networks*, 8(5):997–1007, 1997.

- [49] 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.
- [50] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.
- [51] 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.
- [52] 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.
- [53] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2015.
- [54] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.
- [55] W. Shi, S. Song, and C. Wu. Soft policy gradient method for maximum entropy deep reinforcement learning. *arXiv preprint arXiv:1909.03198*, 2019.
- [56] 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.
- [57] 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.
- [58] 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.
- [59] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS)*, 1996.
- [60] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [61] 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.
- [62] 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.
- [63] C. Tsallis. Possible generalization of Boltzmann-Gibbs statistics. *Journal of statistical physics*, 52(1-2):479–487, 1988.
- [64] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine learning*, 16(3):185–202, 1994.
- [65] 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.
- [66] 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.
- [67] H. van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.
- [68] 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.
- [69] H. van Seijen. Effective multi-step temporal-difference learning for non-linear function approximation. *arXiv preprint arXiv:1608.05151*, 2016.
- [70] 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.
- [71] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
- [72] C. J. C. H. Watkins. Learning from delayed rewards. 1989.
- [73] R. J. Williams. *Reinforcement-learning connectionist systems*. College of Computer Science, Northeastern

- University, 1987.
- [74] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
 - [75] R. J. Williams and J. Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.
 - [76] W. Yang, X. Li, and Z. Zhang. A regularized approach to sparse optimal policy in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5940–5950, 2019.
 - [77] 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.