图 11.10 第三人称视角模仿学习框架<sup>[3]</sup>

## 11.4 序列生成和聊天机器人

我们其实可以把句子生成 (sentence generation) 或聊天机器人理解为模仿学习。如图 11.11 所示，机器在模仿人写句子，我们在写句子的时候，将写下的每一个字都想成一个动作，所有的字合起来就是一个回合。例如，句子生成里面，我们会给机器看很多人类写的字。如果要让机器学会写诗，就要给它看唐诗三百首。人类写的字其实就是专家的示范。每一个词汇其实就是一个动作。机器做句子生成的时候，其实就是在模仿专家的轨迹。聊天机器人也是一样的，在聊天机器人里面我们会收集到很多人交互对话的记录，这些就是专家的示范。

如果我们单纯用最大似然 (maximum likelihood) 这个技术来最大化会得到似然 (likelihood)，这其实就是行为克隆。行为克隆就是看到一个状态，接下来预测我们会得到什么样的动作，有一个标准答案 (ground truth) 告诉机器什么样的动作是最好的。在做似然的时候也是一样的，给定句子已经产生的部分，接下来机器要预测写哪一个字才是最好的。所以，其实最大似然在做序列生成 (sequence generation) 的时候，它对应到模仿学习里面就是行为克隆。只有最大似然是不够的，我们想要用序列生成对抗网络 (sequence GAN)。其实序列生成对抗网络对应逆强化学习，逆强化学习就是一种生成对抗网络的技术。我们把逆强化学习的技术放在句子生成、聊天机器人里面，其实就是序列生成对抗网络与它的种种变形。

句子生成	聊天机器人
专家轨迹:	专家轨迹:
床前明月光	输入: how are you
$(s_1, a_1)$ : ("<BOS>", "床")	$(s_1, a_1)$ : ("输入, <BOS>", "I")
$(s_2, a_2)$ : ("床", "前")	$(s_2, a_2)$ : ("输入, I", "am")
$(s_3, a_3)$ : ("床前", "明")	$(s_3, a_3)$ : ("输入, I am", "fine")
⋮	⋮

图 11.11 模仿学习例子

## 11.5 关键词

模仿学习 (imitation learning, IL)：其讨论我们没有奖励或者无法定义奖励但是有与环境进行交互时怎么进行智能体的学习。这与我们平时处理的问题有些类似，因为通常我们无法从环境中得到明确的奖励。模仿学习又被称为示范学习 (learning from demonstration)、学徒学习 (apprenticeship learning) 以及观察学习 (learning by watching) 等。

行为克隆 (behavior cloning)：类似于机器学习中的监督学习，通过收集专家的状态与动作等对应信息，来训练我们的网络。在使用时，输入状态就可以输出对应的动作。

数据集聚合 (dataset aggregation): 用来应对在行为克隆中专家提供不到数据的情况，其希望收集专家在各种极端状态下的动作。

逆强化学习 (inverse reinforcement learning, IRL): 逆强化学习先找出奖励函数，再用强化学习找出最优演员。这么做是因为我们没有环境中的奖励，但是有专家的示范，使用逆强化学习，我们可以推断专家是因为何种奖励函数才会采取这些动作。有了奖励函数以后就可以使用一般的强化学习方法找出最优演员。

第三人称视角模仿学习 (third person imitation learning): 一种把第三人称视角所观察到的经验泛化为第一人称视角的经验的技术。

## 11.6 习题

**11-1** 具体的模仿学习方法有哪些？

**11-2** 行为克隆存在哪些问题呢？对应的解决方法有哪些？

**11-3** 逆强化学习是怎么运行的呢？

**11-4** 逆强化学习方法与生成对抗网络在图像生成中有什么异曲同工之处？

## 参考文献

- [1] 周志华. 机器学习[M]. 北京: 清华大学出版社, 2016.
- [2] ABBEEL P, DOLGOV D, NG A, et al. Apprenticeship learning for motion planning, with application to parking lot navigation[C]//Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, 2008: 1083-1090.
- [3] STADIE B C, ABBEEL P, SUTSKEVER I. Third-person imitation learning[C]//International Conference on Learning Representations. ICLR, 2017.

## 第 12 章 深度确定性策略梯度

### 12.1 离散动作与连续动作的区别

离散动作与连续动作是相对的概念，一个是可数的，一个是不可数的。如图 12.1 所示，离散动作和连续动作有几个例子。在 *CartPole* 环境中，可以有向左推小车、向右推小车两个动作。在 *Frozen Lake* 环境中，小乌龟可以有上、下、左、右 4 个动作。在雅达利的 *Pong* 游戏中，游戏有 6 个按键的动作可以输出。但在实际情况中，我们经常会遇到连续动作空间的情况，也就是输出的动作是不可数的。比如：推小车推力的大小、选择下一时刻方向盘转动的具体角度、给四轴飞行器的 4 个螺旋桨给的电压的大小。

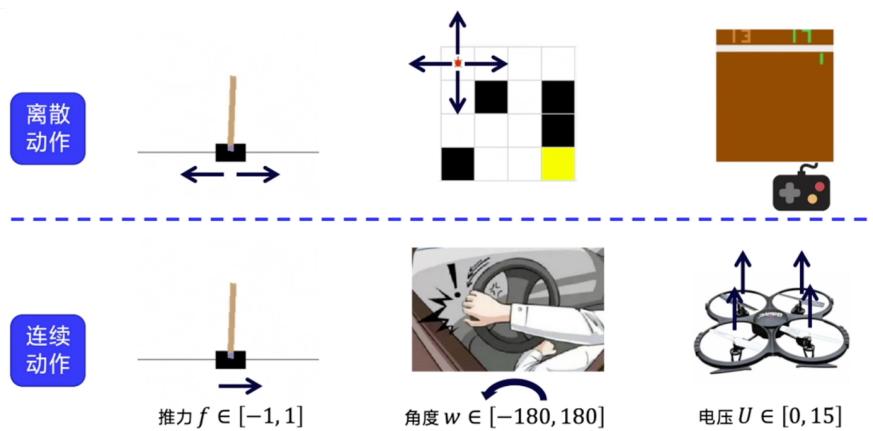


图 12.1 离散动作和连续动作的区别

对于这些连续的动作，Q 学习、深度 Q 网络等算法是没有办法处理的。那我们怎么输出连续的动作呢？这个时候，“万能”的神经网络又出现了。如图 12.2 所示，在离散动作的场景下，比如我们输出上、下或是停止这几个动作。有几个动作，神经网络就输出几个概率值，我们用  $\pi_\theta(a_t|s_t)$  来表示这个随机性的策略。在连续的动作场景下，比如我们要输出机械臂弯曲的角度，我们就输出一个具体的浮点数。我们用  $\mu_\theta(s_t)$  来代表这个确定性的策略。

我们再对随机性策略与确定性策略进行解释。对随机性策略来说，输入某一个状态  $s$ ，采取某一个动作的可能性并不是百分之百的，而是有一个概率的（就好像抽奖一样），根据概率随机抽取一个动作。而对于确定性策略来说，它不受概率的影响。当神经网络的参数固定之后，输入同样的状态，必然输出同样的动作，这就是确定性策略。

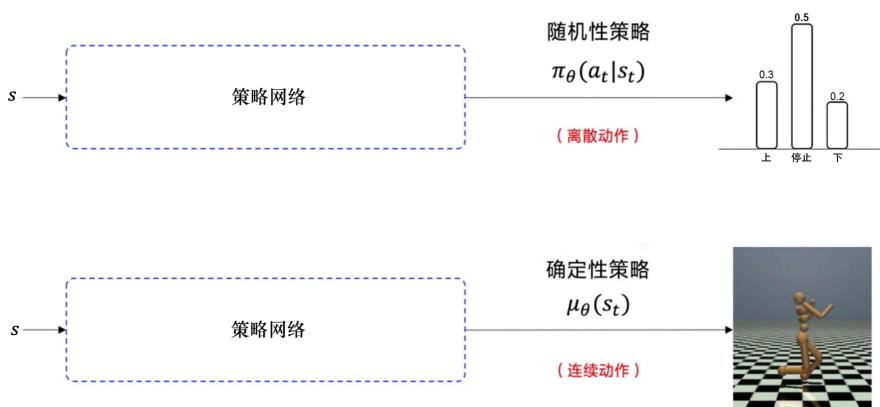


图 12.2 使用神经网络处理连续动作与离散动作

如图 12.3 所示，要输出离散动作，我们就加一个 softmax 层来确保所有的输出是动作概率，并且所有的动作概率和为 1。要输出连续动作，我们一般可以在输出层加一层 tanh 函数。tanh 函数的作用就是

把输出限制到  $[-1,1]$ 。我们得到输出后，就可以根据实际动作的范围将其缩放，再输出给环境。比如神经网络输出一个浮点数 2.8，经过  $\tanh$  函数之后，它就可以被限制在  $[-1,1]$  之间，输出 0.99。假设小车速度的范围是  $[-2,2]$ ，我们就按比例从  $[-1,1]$  扩大到  $[-2,2]$ ，0.99 乘 2，最终输出的就是 1.98，将其作为小车的速度或者推小车的推力输出给环境。

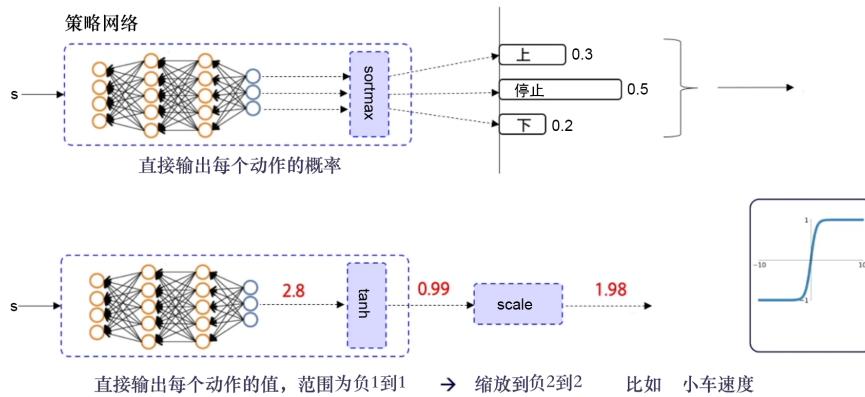


图 12.3 使用神经网络输出离散动作与连续动作

## 12.2 深度确定性策略梯度

在连续控制领域，比较经典的强化学习算法就是深度确定性策略梯度（deep deterministic policy gradient, DDPG）。如图 12.4 所示，DDPG 的特点可以从它的名字中拆解出来，拆解成深度、确定性和策略梯度。

深度是因为用了神经网络；确定性表示 DDPG 输出的是一个确定性的动作，可以用于有连续动作的环境；策略梯度代表的是它用到的是策略网络。REINFORCE 算法每隔一个回合就更新一次，但 DDPG 是每个步骤都会更新一次策略网络，它是一个单步更新的策略网络。

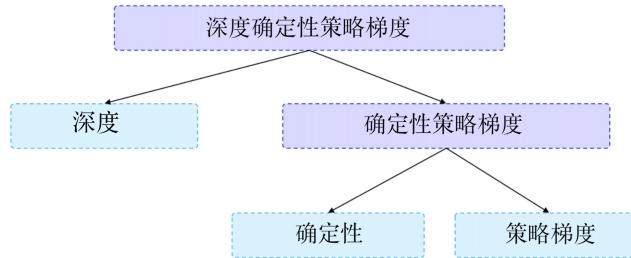


图 12.4 DDPG

DDPG 是深度 Q 网络的一个扩展版本，可以扩展到连续动作空间。在 DDPG 的训练中，它借鉴了深度 Q 网络的技巧：目标网络和经验回放。经验回放与深度 Q 网络是一样的，但目标网络的更新与深度 Q 网络的有点儿不一样。提出 DDPG 是为了让深度 Q 网络可以扩展到连续的动作空间，就是我们刚才提到的小车速度、角度和电压等这样的连续值。如图 12.5 所示，DDPG 在深度 Q 网络基础上加了一个策略网络来直接输出动作值，所以 DDPG 需要一边学习 Q 网络，一边学习策略网络。Q 网络的参数用  $w$  来表示。策略网络的参数用  $\theta$  来表示。我们称这样的结构为演员-评论员的结构。

通俗地解释一下演员-评论员结构。如图 12.6 所示，策略网络扮演的就是演员的角色，它负责对外展示输出，输出动作。Q 网络就是评论员，它会在每一个步骤都对演员输出的动作做一个评估，打一个分，估计演员的动作未来能有多少奖励，也就是估计演员输出的动作的 Q 值大概是多少，即  $Q_w(s, a)$ 。演员需要根据舞台目前的状态来做出一个动作。评论员就是评委，它需要根据舞台现在的状态和演员输出的动

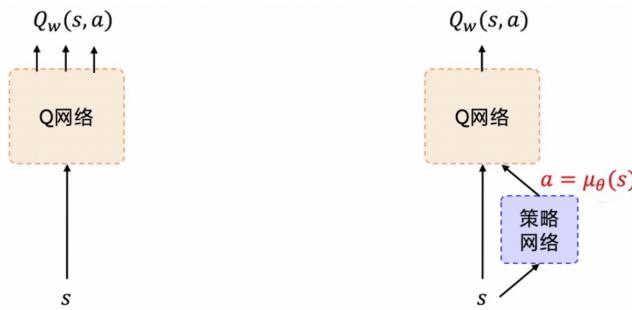


图 12.5 从深度 Q 网络到 DDPG

作对演员刚刚的表现去打一个分数  $Q_w(s, a)$ 。演员根据评委的打分来调整自己的策略，也就是更新演员的神经网络参数  $\theta$ ，争取下次可以做得更好。评论员则要根据观众的反馈，也就是环境的反馈奖励来调整自己的打分策略，也就是要更新评论员的神经网络的参数  $w$ ，它的目标是要让每一场表演都获得观众尽可能多的欢呼声与掌声，也就是要最大化未来的总奖励。

最开始训练的时候，这两个神经网络的参数是随机的。所以评论员最开始是随机打分的，演员也随机输出动作。但是由于有环境反馈的奖励存在，因此评论员的评分会越来越准确，所评判的演员的表现也会越来越好。既然演员是一个神经网络，是我们希望训练好的策略网络，我们就需要计算梯度来更新优化它里面的参数  $\theta$ 。简单来说，我们希望调整演员的网络参数，使得评委打分尽可能高。注意，这里的演员是不关注观众的，它只关注评委，它只迎合评委的打分  $Q_w(s, a)$ 。

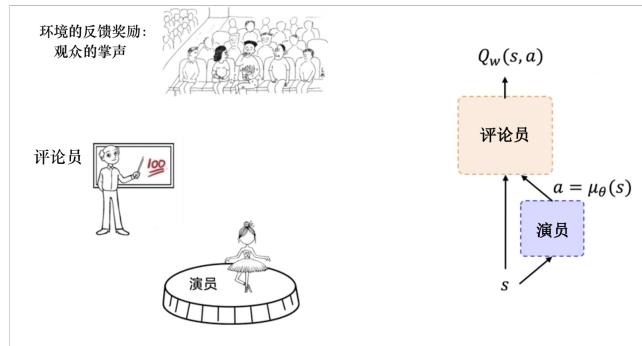


图 12.6 演员-评论员结构通俗解释

深度 Q 网络与 DDPG 的联系如图 12.7 所示。深度 Q 网络的最佳策略是想要学出一个很好的 Q 网络，学出这个网络之后，我们希望选取的那个动作使 Q 值最大。DDPG 的目的也是求解让 Q 值最大的那个动作。演员只是为了迎合评委的打分而已，所以优化策略网络的梯度就是要最大化这个 Q 值，所以构造的损失函数就是让 Q 取一个负号。我们写代码的时候把这个损失函数放入优化器里面，它就会自动最小化损失，也就是最大化 Q。

这里要注意，除了策略网络要做优化，DDPG 还有一个 Q 网络也要优化。评论员一开始也不知道怎么评分，它也是在一步一步的学习当中，慢慢地给出准确的分数。我们优化 Q 网络的方法其实与深度 Q 网络优化 Q 网络的方法是一样的，我们用真实的奖励  $r$  和下一步的  $Q$  即  $Q'$  来拟合未来的奖励  $Q_{\text{target}}$ 。然后让 Q 网络的输出逼近  $Q_{\text{target}}$ 。所以构造的损失函数就是直接求这两个值的均方差。构造好损失函数后，我们将其放到优化器中，让它自动最小化损失。

如图 12.8 所示，我们可以把两个网络的损失函数构造出来。策略网络的损失函数是一个复合函数。我们把  $a = \mu_\theta(s)$  代入，最终策略网络要优化的是策略网络的参数  $\theta$ 。Q 网络要优化的是  $Q_w(s, a)$  和  $Q_{\text{target}}$  之间的一个均方差。但是 Q 网络的优化存在一个和深度 Q 网络一模一样的问题就是它后面的  $Q_{\text{target}}$  是不稳定的。此外，后面的  $Q_{\bar{w}}(s', a')$  也是不稳定的，因为  $Q_{\bar{w}}(s', a')$  也是一个预估的值。

为了使  $Q_{\text{target}}$  更加稳定，DDPG 分别给 Q 网络和策略网络搭建了目标网络，即 target\_Q 网络

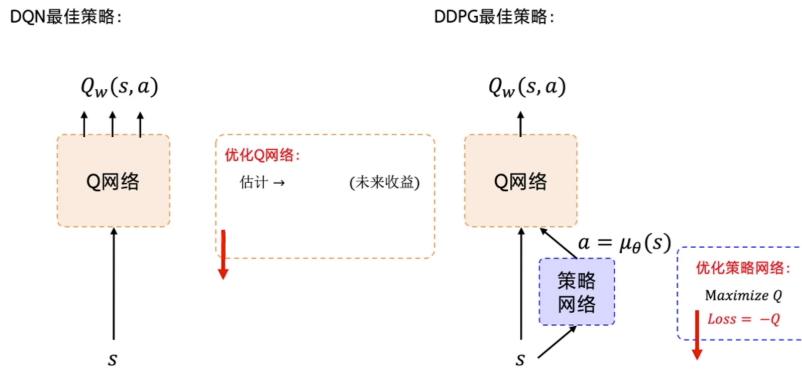


图 12.7 深度 Q 网络与 DDPG 的联系

和 target\_P 策略网络。target\_Q 网络是为了计算  $Q_{\text{target}}$  中  $Q_{\bar{w}}(s', a')$ 。 $Q_{\bar{w}}(s', a')$  里面的需要的下一个动作  $a'$  是通过 target\_P 网络输出的，即  $a' = \mu_{\bar{\theta}}(s')$ 。Q 网络和策略网络的参数是  $w$ ，target\_Q 网络和 target\_P 策略网络的参数是  $\bar{w}$ 。DDPG 有 4 个网络，策略网络的目标网络和 Q 网络的目标网络是颜色比较深的这两个，它们只是为了让计算  $Q_{\text{target}}$  更稳定。因为这两个网络也是固定一段时间的参数之后再与评估网络同步最新的参数。

这里训练需要用到的数据就是  $s$ 、 $a$ 、 $r$ 、 $s'$ ，我们只需要用到这 4 个数据。我们用回放缓冲区把这些数据存起来，然后采样进行训练。经验回放的技巧与深度 Q 网络中的是一样的。注意，因为 DDPG 使用了经验回放技巧，所以 DDPG 是一个异策略的算法。

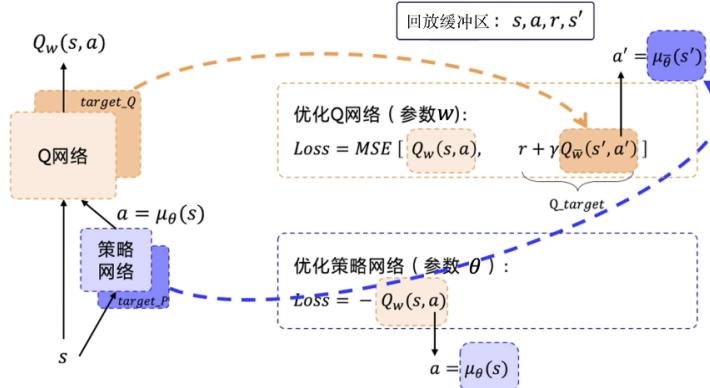


图 12.8 目标网络和经验回放

DDPG 通过异策略的方式来训练一个确定性策略。因为策略是确定的，所以如果智能体使用同策略来探索，在一开始的时候，它很可能不会尝试足够多的动作来找到有用的学习信号。为了让 DDPG 的策略更好地探索，我们在训练的时候给它们的动作加了噪声。DDPG 的原作者推荐使用时间相关的 OU 噪声，但最近的结果表明不相关的、均值为 0 的高斯噪声的效果非常好。由于后者更简单，因此我们更喜欢使用它。为了便于获得更高质量的训练数据，我们可以在训练过程中把噪声变小。在测试的时候，为了查看策略利用它学到的东西的表现，我们不会在动作中加噪声。

### 12.3 双延迟深度确定性策略梯度

虽然 DDPG 有时表现很好，但它对于超参数和其他类型的调整方面经常很敏感。如图 12.9 所示，DDPG 常见的问题是已经学习好的 Q 函数开始显著地高估 Q 值，然后导致策略被破坏，因为它利用了 Q

函数中的误差。

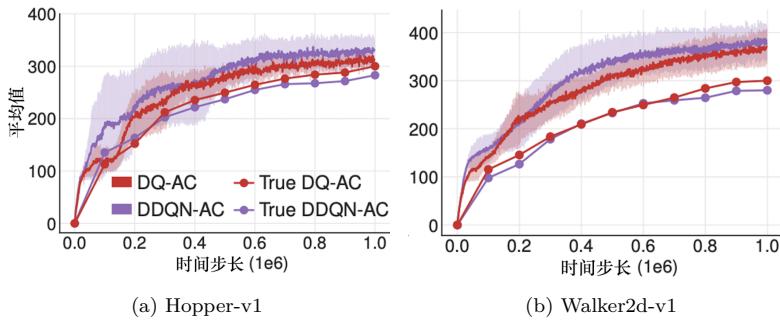


图 12.9 DDPG 的问题<sup>[1]</sup>

我们可以使用实际的 Q 值与 Q 网络输出的 Q 值进行对比。实际的 Q 值可以用蒙特卡洛来算。根据当前的策略采样 1000 条轨迹，得到  $G$  后取平均值，进而得到实际的 Q 值。

**双延迟深度确定性策略梯度 (twin delayed DDPG, TD3)** 通过引入 3 个关键技巧来解决这个问题。

- **截断的双 Q 学习 (clipped double Q-learning)**。TD3 学习两个 Q 函数（因此名字中有“twin”）。TD3 通过最小化均方差来同时学习两个 Q 函数： $Q_{\phi_1}$  和  $Q_{\phi_2}$ 。两个 Q 函数都使用一个目标，两个 Q 函数中给出的较小的值会被作为如下的 Q-target：

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_i, \text{targ}}(s', a_{\text{TD3}}(s'))$$

- **延迟的策略更新 (delayed policy updates)**。相关实验结果表明，同步训练动作网络和评价网络，却不使用目标网络，会导致训练过程不稳定；但是仅固定动作网络时，评价网络往往能够收敛到正确的结果。因此 TD3 算法以较低的频率更新动作网络，以较高的频率更新评价网络，通常每更新两次评价网络就更新一次策略。
  - **目标策略平滑 (target policy smoothing)**。TD3 引入了平滑化 (smoothing) 思想。TD3 在目标动作中加入噪声，通过平滑 Q 沿动作的变化，使策略更难利用 Q 函数的误差。
- 这 3 个技巧加在一起，使得性能相比基线 DDPG 有了大幅的提升。

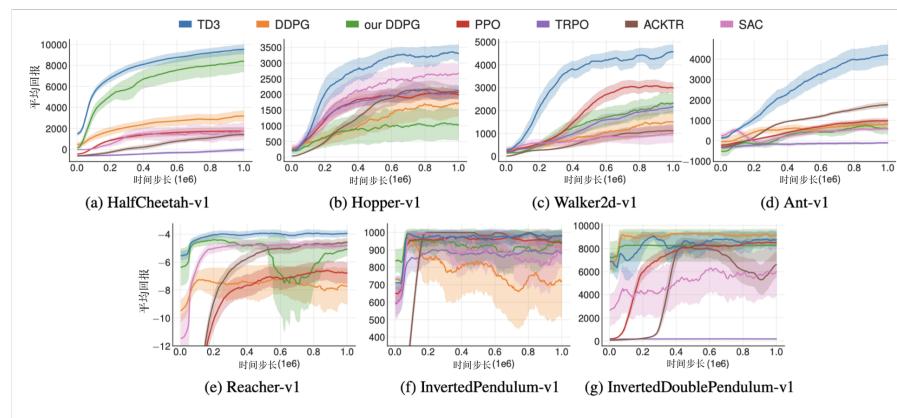
目标策略平滑化的工作原理如下：

$$a_{\text{TD3}}(s') = \text{clip}(\mu_{\theta, \text{targ}}(s') + \text{clip}(\epsilon, -c, c), a_{\text{low}}, a_{\text{high}})$$

其中  $\epsilon$  本质上是一个噪声，是从正态分布中取样得到的，即  $\epsilon \sim N(0, \sigma)$ 。目标策略平滑化是一种正则化方法。

如图 12.10 所示，我们可以将 TD3 算法与其他算法进行对比。TD3 算法的作者自己实现的深度确定性策略梯度 (图中为 our DDPG) 和官方实现的 DDPG 的表现不一样，这说明 DDPG 对初始化和调参非常敏感。TD3 对参数不是这么敏感。在 TD3 的论文中，TD3 的性能比软演员-评论员 (soft actor-critic, SAC) 高。软演员-评论员又被译作软动作评价。但在 SAC 的论文中，SAC 的性能比 TD3 高，这是因为强化学习的很多算法估计对参数和初始条件敏感。

TD3 的作者给出了其对应 PyTorch 的实现，见 <https://github.com/sfujim/TD3/>，代码写得很棒，我们可以将其作为一个强化学习的标准库来学习。TD3 以异策略的方式训练确定性策略。由于该策略是确定性的，因此如果智能体要探索策略，则一开始它可能不会尝试采取足够广泛的动作来找到有用的学习信号。为了使 TD3 策略更好地探索，我们在训练时在它们的动作中添加了噪声，通常是不相关的均值为 0 的高斯噪声。为了便于获取高质量的训练数据，我们可以在训练过程中减小噪声的大小。在测试时，为了查看策略对所学知识的利用程度，我们不会在动作中增加噪声。

图 12.10 TD3 与其他算法对比<sup>[1]</sup>

## 12.4 使用深度确定性策略梯度解决倒立摆问题

前面章节都是离散动作的环境，但实际中也有很多连续动作的环境，比如 OpenAI Gym 中的 Pendulum-v0 环境，它解决的是一个倒立摆问题，我们先对该环境做一个简要说明。

### 12.4.1 Pendulum-v0 简介

如果说 CartPole-v0 是一个离散动作的经典入门环境，那么 Pendulum-v0 就是连续动作的经典入门环境。如图 12.11 所示，我们通过施加力矩使摆阵向上摆动并保持直立。

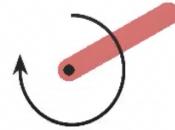


图 12.11 Pendulum-v0 环境

该环境的状态数有三个，设摆针在竖直方向上的顺时针旋转角为  $\theta$ ,  $\theta$  设在  $[\pi, \pi]$  之间，则相应的状态为  $[\cos\theta, \sin\theta, \dot{\theta}]$ ，即表示角度和角速度。我们的动作则是一个 -2 到 2 之间的力矩，它是一个连续量，因而该环境不能用离散动作的算法比如深度 Q 网络算法来解决。奖励是根据相关的物理原理而计算出的等式，如下：

$$-\left(\theta^2 + 0.1 \times \dot{\theta}^2 + 0.001 \times \text{action}^2\right)$$

对于每一步，其最低奖励为  $-(\pi^2 + 0.1 \times 8^2 + 0.001 \times 2^2) = -16.2736044$ ，最高奖励为 0。同 CartPole-v0 环境一样，达到最优算法的情况下，每回合的步数是无限的，因此这里设定每回合最大步数为 200 以便于训练。

### 12.4.2 深度确定性策略梯度基本接口

我们依然使用接口的概念，通过伪代码分析并实现 DDPG 的训练模式，如图 12.12 所示。

代码如下：

```

ou_noise = OUNoise(env.action_space) # 动作噪声
rewards = [] # 记录奖励
ma_rewards = [] # 记录滑动平均奖励
for i_ep in range(cfg.train_eps):
    state = env.reset()

```

- 初始化评论员网络  $Q(s, a|\theta^Q)$  和演员网络  $\mu(s|\theta^\mu)$ , 其权重分别为  $\theta^Q$  和  $\theta^\mu$
- 初始化目标  $Q'$  和  $\mu'$ , 并复制权重  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
- 初始化经验回放缓冲区  $R$
- 执行  $M$  个回合循环, 对于每个回合
  - 初始化动作探索的随机过程, 即噪声  $\mathcal{N}$
  - 初始化状态  $s_i$
  - 循环  $T$  个时间步长, 对于每个时步  $t$ 
    - 根据当前的策略和噪声选择动作  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
    - 环境根据  $a_t$  反馈奖励  $r_t$  和下一个状态  $s_{t+1}$
    - 存储转移即  $(s_t, a_t, r_t, s_{t+1})$  到经验回放  $D$  中
    - 更新策略如下:
      - 1. 从  $D$  中随机采样  $N$  个小批量的转移
      - 2. 计算实际的  $Q$  值  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
      - 3. 对损失函数  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$  关于参数  $\theta$  做随机梯度下降更新评论员网络
      - 4. 使用采样梯度更新演员网络:
 
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$
      - 5. 软更新目标网络:
 
$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}, \theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

图 12.12 深度确定性策略梯度算法

```

ou_noise.reset()
done = False
ep_reward = 0
i_step = 0
while not done:
    i_step += 1
    action = agent.choose_action(state)
    action = ou_noise.get_action(action, i_step)
    next_state, reward, done, _ = env.step(action)
    ep_reward += reward
    agent.memory.push(state, action, reward, next_state, done)
    agent.update()
    state = next_state
    if (i_ep+1)%10 == 0:
        print('回合: {}/{}, 奖励: {}'.format(i_ep+1, cfg.train_eps, ep_reward))
    rewards.append(ep_reward)
    if ma_rewards:
        ma_rewards.append(0.9*ma_rewards[-1]+0.1*ep_reward)
    else:
        ma_rewards.append(ep_reward)

```

相比于深度  $Q$  网络, DDPG 主要更新了两部分, 一是给动作施加噪声, 另外是软更新策略, 即最后一步。

#### 12.4.3 Ornstein-Uhlenbeck 噪声

奥恩斯坦-乌伦贝克 (Ornstein-Uhlenbeck, OU) 噪声适用于惯性系统, 尤其是时间离散化粒度较小的情况。OU 噪声是一种随机过程, 下面略去证明, 直接给出公式。对于当前时刻  $t$  的一个变量  $x$ , 其下一时刻  $x(t + \Delta t)$ :

$$x(t + \Delta t) = x(t) - \theta(x(t) - \mu)\Delta t + \sigma W_t$$

其中  $W_t$  属于正态分布，OU 噪声代码实现如下：

```
class OUNoise( object):
    '''Ornstein-Uhlenbeck噪声
    ...
    def __init__(self, action_space, mu=0.0, theta=0.15, max_sigma=0.3, min_sigma=0.3, decay_period
                 =100000):
        self.mu          = mu # OU噪声的参数
        self.theta       = theta # OU噪声的参数
        self.sigma       = max_sigma # OU噪声的参数
        self.max_sigma   = max_sigma
        self.min_sigma   = min_sigma
        self.decay_period = decay_period
        self.action_dim  = action_space.shape[0]
        self.low          = action_space.low
        self.high         = action_space.high
        self.reset()
    def reset(self):
        self.obs = np.ones(self.action_dim) * self.mu
    def evolve_obs(self):
        x = self.obs
        dx = self.theta * (self.mu - x) + self.sigma * np.random.randn(self.action_dim)
        self.obs = x + dx
        return self.obs
    def get_action(self, action, t=0):
        ou_obs = self.evolve_obs()
        self.sigma = self.max_sigma - (self.max_sigma - self.min_sigma) *
            min(1.0, t / self.decay_period) # sigma会逐渐衰减
        return np.clip(action + ou_obs, self.low, self.high) # 动作加上噪声后进行剪切
```

#### 12.4.4 深度确定性策略梯度算法

DDPG 算法主要也包括两个功能，一个是选择动作，另外一个是更新策略，首先看选择动作：

```
def choose_action(self, state):
    state = torch.FloatTensor(state).unsqueeze(0).to(self.device)
    action = self.actor(state)
    return action.detach().cpu().numpy()[0, 0]
```

由于 DDPG 是直接从演员网络取得动作，因此这里不用  $\varepsilon$ -贪心策略。在更新策略函数中，也会与深度 Q 网络稍有不同，并且加入软更新：

```
def update(self):
    if len(self.memory) < self.batch_size: # 当 memory 中不满足一个批量时，不更新策略
        return
    # 从回放缓冲区中随机采样一个批量的经验
    state, action, reward, next_state, done = self.memory.sample(self.batch_size)
    # 转变为张量
    state = torch.FloatTensor(state).to(self.device)
    next_state = torch.FloatTensor(next_state).to(self.device)
    action = torch.FloatTensor(action).to(self.device)
    reward = torch.FloatTensor(reward).unsqueeze(1).to(self.device)
    done = torch.FloatTensor(np.float32(done)).unsqueeze(1).to(self.device)
```

```

# 计算期望Q值
policy_loss = self.critic(state, self.actor(state))
policy_loss = -policy_loss.mean()
next_action = self.target_actor(next_state)
target_value = self.target_critic(next_state, next_action.detach())
expected_value = reward + (1.0 - done) * self.gamma * target_value
expected_value = torch.clamp(expected_value, -np.inf, np.inf)
value = self.critic(state, action)
value_loss = nn.MSELoss()(value, expected_value.detach())
self.actor_optimizer.zero_grad()
policy_loss.backward()
self.actor_optimizer.step()
self.critic_optimizer.zero_grad()
value_loss.backward()
self.critic_optimizer.step()
# 软更新
for target_param, param in zip(self.target_critic.parameters(), self.critic.parameters()):
    target_param.data.copy_(target_param.data * (1.0 - self.soft_tau) + param.data * self.soft_tau)
for target_param, param in zip(self.target_actor.parameters(), self.actor.parameters()):
    target_param.data.copy_(target_param.data * (1.0 - self.soft_tau) + param.data * self.soft_tau)

```

#### 12.4.5 结果分析

实现算法之后，我们先看看训练效果，如图 12.13 所示。

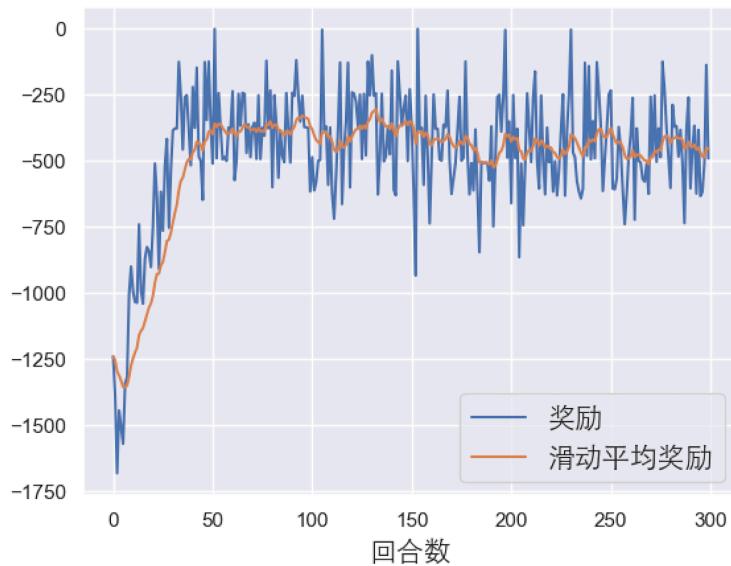


图 12.13 Pendulum-v0 环境下 DDPG 算法的训练曲线

可以看到算法整体上是收敛的，但是稳定状态下波动还比较大，依然有提升的空间。限于笔者的精力，这里只是帮助读者实现一个基础的代码演示，想要使得算法调到最优，感兴趣的读者可以多思考实现。我们再来看看测试的结果，如图 12.14 所示。

从图 12.14 中看出测试的滑动平均奖励在 -150 左右，但其实训练的时候平均的稳态奖励在 -300 左右，这是因为在测试的时候我们舍去了 OU 噪声。

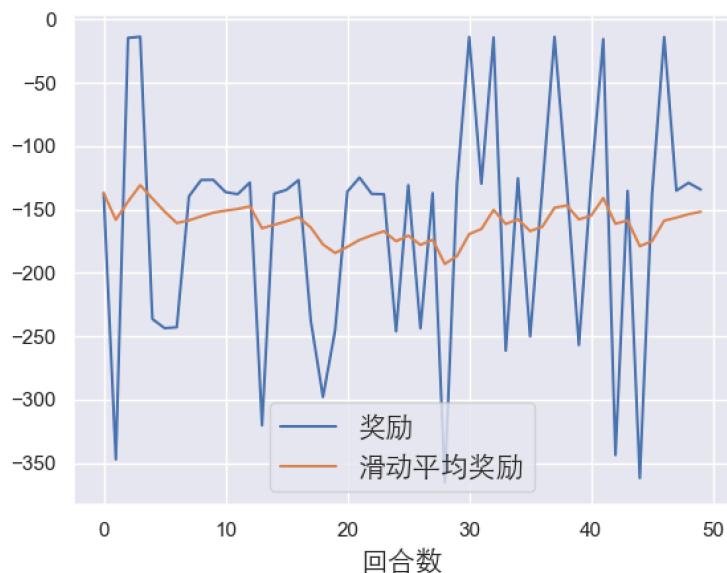


图 12.14 Pendulum-v0 环境下 DDPG 算法的测试曲线

## 12.5 关键词

深度确定性策略梯度 (deep deterministic policy gradient, DDPG): 在连续控制领域经典的强化学习算法，是深度 Q 网络在处理连续动作空间的一个扩充方法。具体地，从命名就可以看出，“深度”表明使用了深度神经网络；“确定性”表示其输出的是一个确定的动作，可以用于连续动作环境；“策略梯度”代表的是它用到的是策略网络，并且每步都会更新一次，其是一个单步更新的策略网络。其与深度 Q 网络都有目标网络和经验回放的技巧，在经验回放部分是一致的，在目标网络的更新上有些许不同。

## 12.6 习题

**12-1** 请解释随机性策略和确定性策略，两者有什么区别？

**12-2** 对于连续动作的控制空间和离散动作的控制空间，如果我们都采取策略网络，应该分别如何操作？

## 12.7 面试题

**12-1** 友善的面试官：请简述一下深度确定性策略梯度算法。

**12-2** 友善的面试官：请问深度确定性策略梯度算法是同策略算法还是异策略算法？请说明具体原因并分析。

**12-3** 友善的面试官：你是否了解过分布的分布式深度确定性策略梯度算法 (distributed distributional deep deterministic policy gradient, D4PG) 呢？请描述一下吧。

## 参考文献

- [1] FUJIMOTO S, HOOF H, MEGER D. Addressing function approximation error in actor-critic methods [C]//International Conference on Machine Learning. PMLR, 2018: 1587-1596.

## 第 13 章 AlphaStar 论文解读

### 13.1 AlphaStar 以及背景简介

相比于之前的深蓝和 AlphaGo，对于《星际争霸 II》等策略对战型游戏，使用 AI 与人类对战的难度更大。比如在《星际争霸 II》中，要想在玩家对战玩家的模式中击败对方，就要学会各种战术，各种微操和掌握时机。在游戏中玩家还需要对对方阵容的更新实时地做出正确判断以及行动，甚至要欺骗对方以达到战术目的。总而言之，想要让 AI 上手这款游戏是非常困难的。但是 DeepMind 做到了。

AlphaStar 是 DeepMind 与暴雪使用深度强化学习技术实现的计算机与《星际争霸 II》人类玩家对战的产品，其因为近些年在《星际争霸 II》比赛中打败了职业选手以及 99.8% 的欧服玩家而被人所熟知。北京时间 2019 年 1 月 25 日凌晨 2 点，暴雪公司与 DeepMind 合作研发的 AlphaStar 正式通过直播亮相。按照直播安排，AlphaStar 与两位《星际争霸 II》人类职业选手进行了 5 场比赛对决演示。加上并未在直播中演示的对决，在人类对阵 AlphaStar 的共计 11 场比赛中，人类仅取得了 1 场胜利。DeepMind 也将研究工作发表在了 2019 年 10 月的 *Nature* 杂志上。本章将对这篇论文进行深入的分析，有兴趣的读者可以阅读原文。

### 13.2 AlphaStar 的模型输入和输出是什么呢？——环境设计

构建深度强化学习模型的第一步就是构建模型的输入和输出，对于《星际争霸 II》这一个复杂的环境，文章第一步就是将游戏的环境抽象成众多独立的数据信息。

#### 13.2.1 状态（网络的输入）

AlphaStar 将《星际争霸 II》的环境状态分为 4 部分，分别为实体（entities）信息、地图（map）信息、玩家数据（player data）信息、游戏统计（game statistics）信息，如图 13.1 所示。

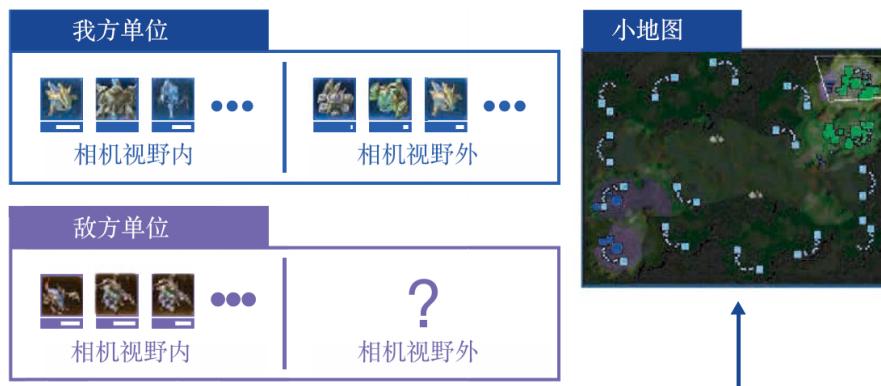


图 13.1 环境状态信息<sup>[1]</sup>

第一部分——实体信息，例如当前时刻环境中有什么建筑、兵种等，并且我们将每一个实体的属性信息使用向量表示。例如对于一个建筑，其当前时刻的向量中包含此建筑的血量、等级、位置以及冷却时间等信息。所以对于当前帧的全部实体信息，环境会给神经网络  $N$  个长度为  $K$  的向量，分别表示此刻智能体能够看见的  $N$  个实体的具体信息（向量信息）。

第二部分——地图信息，这部分比较好理解，即将地图中的信息以矩阵的形式输入神经网络中，来表示当前状态全局地图的信息（向量信息或图像信息）。

第三部分——玩家数据信息，也就是当前状态下，玩家的等级和种族等信息（标量信息）。

第四部分——游戏统计信息，视野的位置（小窗口的位置，区别于第二部分的全局地图信息），还有当前游戏的开始时间等信息（标量信息）。

### 13.2.2 动作（网络的输出）

AlphaStar 的动作信息主要分为 6 个部分，如图 13.2 所示，分别为动作类型（action type）、选中的单元（selected units）、目标（target）、执行动作的队列（queued）、是否重复（repeat）以及延时（delay），各个部分间是有关联的。

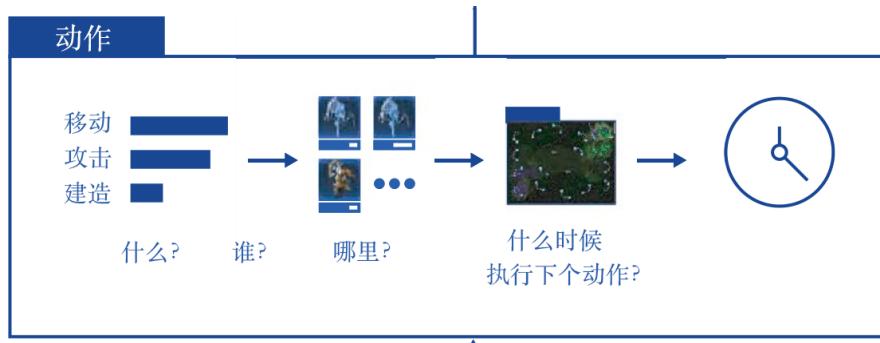


图 13.2 动作信息<sup>[1]</sup>

第一部分——动作类型，即下一次要进行的动作的类型是移动小兵、升级建筑还是移动小窗口的位置等。

第二部分——选中的单元，承接第一部分，例如我们要进行的动作类型是移动小兵，那么我们就应该选择具体移动哪一个小兵。

第三部分——目标，承接第二部分，我们移动小兵 A 后，是要去地图的某一个位置还是去攻击对手的哪一个目标等，即选择目的地或攻击的对象。

第四部分——执行动作的队列，即是否立即执行动作，对于小兵 A，是到达目的地后直接进行攻击还是原地待命。

第五部分——是否重复，如果需要小兵 A 持续攻击，那么就不需要再通过网络计算得到下一个动作，直接重复上一个动作即可。

第六部分——延时，即等候多久后再接收网络的输入，可以理解为一个操作的延迟。

### 13.3 AlphaStar 的计算模型是什么呢？——网络结构

我们在 13.2 节说明了 AlphaStar 网络的输入和输出，即状态和动作，那么从状态怎么得到动作呢？这里我们先给出其网络结构的总览，如图 13.3 所示，后面对此详细讨论。

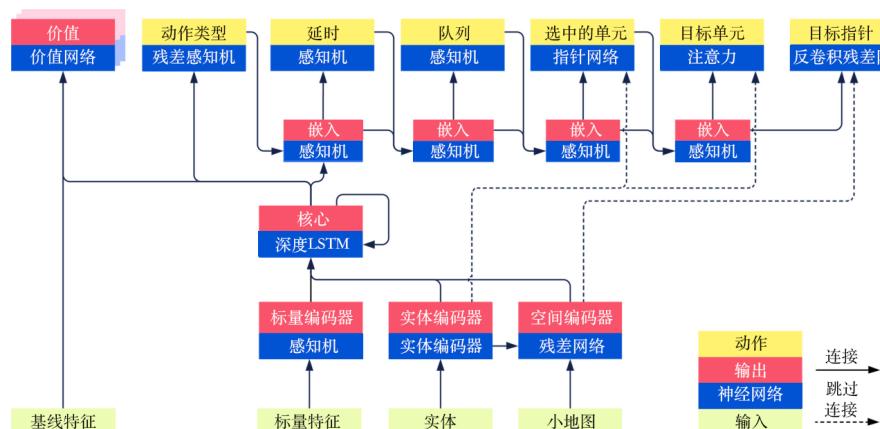


图 13.3 AlphaStar 网络结构总览<sup>[1]</sup>

### 13.3.1 输入部分

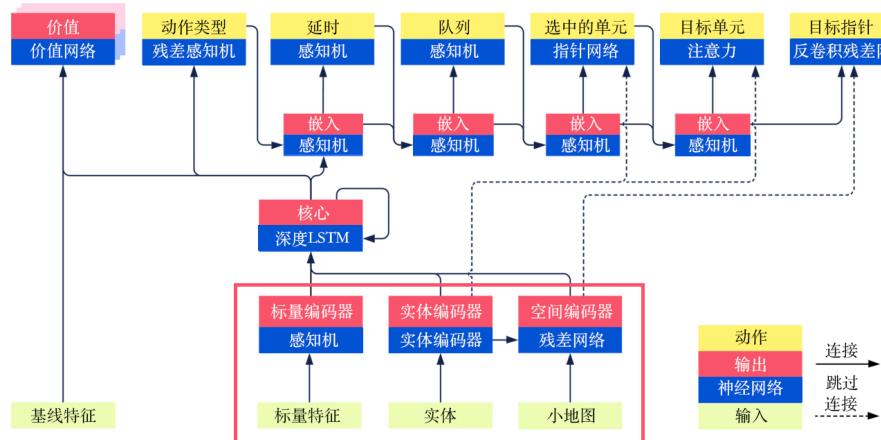


图 13.4 AlphaStar 网络结构输入部分<sup>[1]</sup>

从图 13.4 中的红框可以看出，模型的输入部分主要有 3 个部分：标量特征（scalar features），例如前面描述的玩家等级以及小窗口的位置等信息；实体（entities），是向量，即前面所叙述的一个建筑或一个小兵的当前所有的属性信息；小地图（minimap），即图像数据。

- 对于标量特征，使用多层感知机（multilayer perceptron, MLP），就可以得到对应的向量，可以认为是一个嵌入过程。
- 对于实体，使用自然语言处理中常用的 Transformer 架构作为编码器（encoder）。
- 对于小地图，使用图像中常用的 ResNet 架构作为编码器，得到一个定长的向量。

### 13.3.2 中间过程

中间过程比较简单，即通过一个深度长短期记忆网络模块融合 3 种当前状态下的嵌入并进行下一时刻的输出，如图 13.5 所示，并且将该输出分别送入价值网络（value network）、残差多层感知机（residual MLP）以及动作类型的后续的多层感知机中。

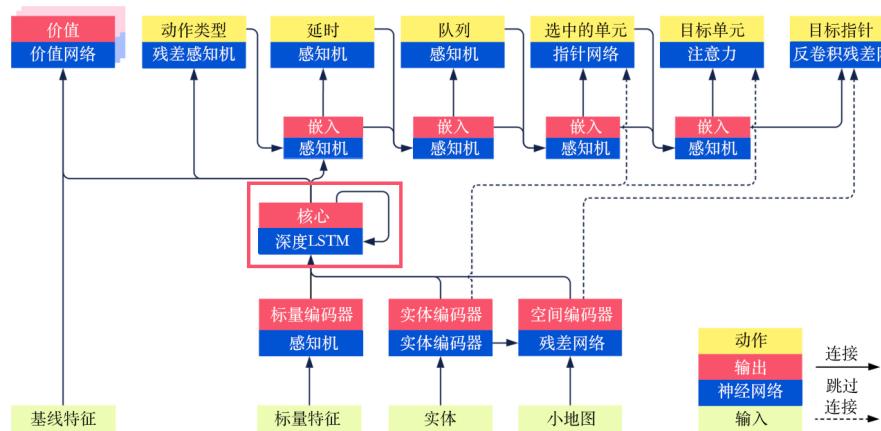


图 13.5 AlphaStar 网络结构中的深度长短期记忆网络模块<sup>[1]</sup>

### 13.3.3 输出部分

正如前面介绍的，输出的动作是前后相关联的，如图 13.6 所示，我们按照顺序一一介绍。

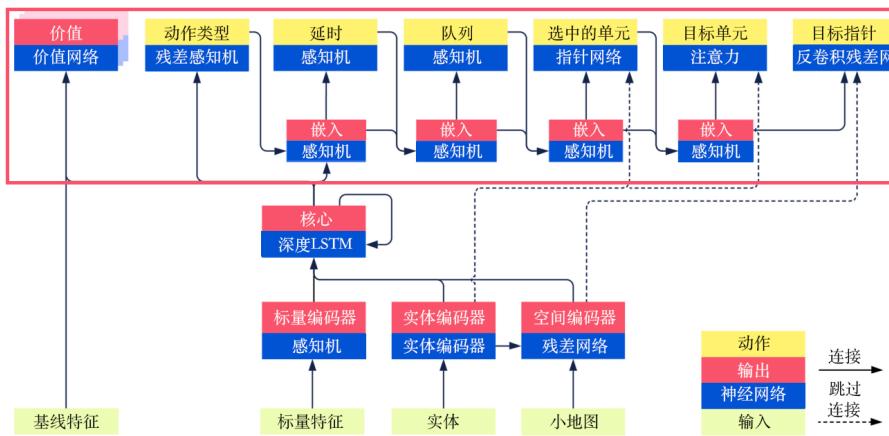


图 13.6 AlphaStar 网络结构输出部分<sup>[1]</sup>

首先是动作类型：使用深度长短期记忆网络的嵌入向量作为输入，使用残差多层感知机得到动作类型的 Softmax 激活函数的输出结果，并将其传给下一个子模型进行嵌入。

然后是延时：将动作类型嵌入的结果以及深度长短期记忆网络的结果一起输入多层感知机后得到结果，并传给下一个子模型进行嵌入。

接下来是执行动作的队列：将延时的结果以及嵌入的结果一起输入多层感知机后得到结果，并传给下一个子模型进行嵌入。

然后是选中的单元：将队列的结果、嵌入的结果以及实体编码后的全部结果（非平均的结果）一起送入指针网络（pointer network）中得到结果，并传给下一个子模型进行嵌入。这里的指针网络的输入是一个序列，输出是另外一个序列，并且输出序列的元素来自输入的序列。其主要用于自然语言处理中，在这里很适合我们选中的单元的计算。

接着是目标单元（target unit）和目标指针（target point）两者二选一，对于目标单元，使用注意力（attention）机制得到最优的动作作用的一个对象；对于目标区域，使用反卷积残差网络，将嵌入的向量反卷积为地图的大小，从而执行目标移动到某一点的对应动作。

## 13.4 庞大的 AlphaStar 如何训练呢？——学习算法

对于上面复杂的模型，AlphaStar 究竟如何进行训练呢？总结下来一共分为 4 个部分，即监督学习（主要是解决训练的初始化问题）、强化学习、模仿学习（配合强化学习）以及多智能体学习或自学习（面向对战的具体问题），下面我们一一分析。

### 13.4.1 监督学习

在训练一开始，AlphaStar 首先使用监督学习即利用人类的数据进行比较好的初始化。模型的输入是收集到的人类的对局数据，输出是训练好的神经网络。具体的做法是，对于收集到的人类对局数据，即对于每一个时刻解码游戏的状态，将每一时刻的状态送入网络中得到每一个动作的概率分布，最终计算模型的输出以及人类对局数据的 KL 散度，并以 KL 散度进行网络的优化，其中在 KL 散度中需要使用不同的损失函数。例如，动作类型的损失，即分类问题的损失就需要使用交叉熵。而对于目标定位等类似的问题就需要计算均方误差（mean square error, MSE）。当然还有一些细节，大家可以自行阅读论文。总之，经过监督学习，模型输出的概率分布就可以与人类玩家输出的概率分布类似。

### 13.4.2 强化学习

这里的目标就是通过优化策略使得期望的奖励最大，即

$$J(\pi_\theta) = E_{\pi_\theta} \sum_{t=0} r(s_t, a_t) \quad (13.1)$$

但 AlphaStar 的训练模型使用非采样模型，即免策略的模型，这是因为其使用的架构为类似于 IMPALA 的架构，即演员负责与环境交互并采样，学习者负责优化网络并更新参数，而演员和学习者通常是异步进行计算的，并且由于前面介绍的输出动作的类型空间复杂，因此导致价值函数的拟合比较困难。

AlphaStar 利用以下的方式进行强化学习模型的构建。

(1) 首先是采取经典的演员-评论员 (actor-critic) 结构，使用策略网络给出当前状态下的智能体的动作，即计算  $\pi(a_t|s_t)$ ，使用价值网络计算当前状态下的智能体的期望奖励，即计算  $V(s_t) = E \sum_{t'=t} r_{t'} = E_{a_t}[r(s_t, a_t) + V(s_{t+1})]$ 。具体的计算方法是：对于当前的状态  $s$ ，计算当前动作  $a$  相对于“平均动作”所能额外获得的奖励。

$$A(s_t, a_t) = [r(s_t, a_t) + V(s_{t+1})] - V(s_t) \quad (13.2)$$

式 (13.2) 即当前动作的预期奖励减去当前状态的预期奖励。在 AlphaStar 中，向上移动的策略更新 (upgoing policy update, UPGO) 也得到了应用，向上移动的策略更新使用一个迭代变量  $G_t$  来取代原来的动作的预期奖励  $r(s_t, a_t) + V(s_{t+1})$ ，即把未来乐观的信息纳入额外奖励中，式 (13.2) 可改写为：

$$A(s_t, a_t) = G_t - V(s_t) \quad (13.3)$$

其中，

$$G_t = \begin{cases} r_t + G_{t+1} & , \text{如果 } Q(s_{t+1}, a_{t+1}) \geq V(s_{t+1}) \\ r_t + V(s_{t+1}) & , \text{否则} \end{cases}$$

(2) 基于上面计算得到的动作，更新策略梯度，即  $\nabla_\theta J = A(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)$ 。我们在前面介绍了，如果基于  $\pi_\theta$  的分布不好求解，或者说学习策略  $\pi_\theta$  与采集策略  $\pi_\mu$  不同，我们需要使用重要性采样，即  $\nabla_\theta J = E_{\pi_\mu} \frac{\pi_\theta(a_t|s_t)}{\pi_\mu(a_t|s_t)} A_{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)$ 。当然我们还需防止  $\frac{\pi_\theta(a_t|s_t)}{\pi_\mu(a_t|s_t)}$  出现无穷大的情况，我们需要使用 V-trace 限制重要性系数。这也是用于免策略的一个更新方法，在 IMPALA 论文中的 4.1 节有所体现。即将重要性系数的最大值限制为 1，公式如下。

$$\nabla_\theta J = E_{\pi_\mu} \rho_t A_{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \quad (13.4)$$

其中，

$$\rho_t = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_\mu(a_t|s_t)}, 1\right)$$

(3) 利用时序差分 ( $\lambda$ ) 来优化价值网络，并同时输入对手的数据。对于我们的价值函数

$$V_{\pi_\theta}(s_t) = E_{\pi_\theta} \sum_{t'=t} \gamma^{t'-t} r(s_t, a_t) = E_{a_t \sim \pi_\theta(\cdot|s_t)} [r(s_t, a_t) + \gamma V(s_{t+1})]$$

可以使用时序差分方法计算均方差损失，有如下几种。

- TD(0)，表达式为  $L = [(r_t + \gamma V_{t+1}) - V_t]^2$ ，即当前步 (step) 的信息，有偏小的方差存在。
- TD(1)，即蒙特卡洛方法，表达式为  $L = [(\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}) - V_t]^2$ ，即未来无穷步的信息，无偏大方差。
- TD( $\lambda$ )，即以上两个方法的加权平均。平衡当前步、下一步到无穷步后的结果。
  - 已知对于  $\lambda \in (0, 1)$ ， $(1-\lambda) + (1-\lambda)\lambda + (1-\lambda)\lambda^2 + \dots = 1$ 。
  - $r_t = \lim_{T \rightarrow \infty} (1-\lambda)(r_t + V_{t+1}) + (1-\lambda)\lambda(r_t + \gamma r_{t+1} + \gamma^2 V_{t+2}) + \dots$

### 13.4.3 模仿学习

模仿学习额外引入了监督学习损失以及人类的统计量  $Z$ ，即对于建造顺序 (build order)、建造单元 (build unit)、升级 (upgrade)、技能 (effect) 等信息进行奖励，并将统计量  $Z$  输入策略网络和价值网络。另外，AlphaStar 对于人类数据的利用还体现在前面介绍的使用监督学习进行网络的预训练工作中。

### 13.4.4 多智能体学习/自学习

自学习在 AlphaGo 中得到了应用。自学习通俗讲就是自己和自己玩，自己和自己对战。AlphaStar 对此进行了一些更新，即有优先级的虚拟自学习策略。虚拟自学习就是在训练过程中，每隔一段时间就进行存档，并随机均匀地从存档中选出对手与正在训练的智能体对战。而有优先级的虚拟自学习指的是优先挑选常能打败智能体的对手进行训练对战，评判指标就是概率。在 AlphaStar 中，其训练的智能体分为 3 种：主智能体（main agent）、联盟利用者（league exploiter）和主利用者（main exploiter）。

**主智能体：**正在训练的智能体及其祖先。其有 0.5 的概率从联盟中的所有对手中挑选对手，使用有优先级的虚拟自学习策略，即能打败智能体的概率高，不能打败智能体的概率低。有 0.35 的概率与自己对战，有 0.15 的概率与能打败智能体的联盟利用者或者先前的智能体对战。

**联盟利用者：**能打败联盟中的所有智能体。其按照有优先级的虚拟自学习策略计算的概率与全联盟的对手训练，在以 0.7 的胜率打败所有的智能体或者距离上次存档  $2 \times 10^9$  步后就保存策略，并且在存档的时候，有 0.25 概率把场上的联盟利用者的策略重设成监督学习初始化的策略。

**主利用者：**能打败训练中的所有智能体。在训练的过程中，随机从 3 个智能体中挑选 1 个主智能体，如果可以以高于 0.1 的概率打败该智能体就与其进行训练，如果不能就从之前的主智能体中再挑选对手。当以 0.7 的胜率打败全部 3 个正在学习的主智能体时，或者距上次存档  $4 \times 10^9$  步之后就保存策略，并且进行重设初始化策略的操作。

他们的区别在于：如何选取训练过程中对战的对手；在什么情况下存档（snapshot）现在的策略；以多大的概率将策略参数重设为监督学习给出的初始化参数。

## 13.5 AlphaStar 实验结果如何呢？——实验结果

### 13.5.1 宏观结果

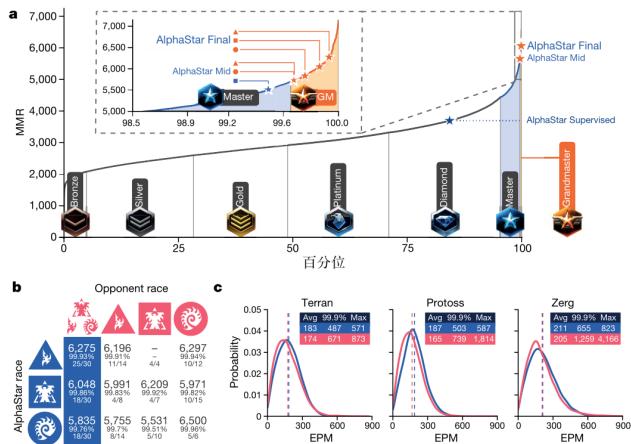


图 13.7 AlphaStar 的实验结果<sup>[1]</sup>

图 13.7(a) 为训练后的智能体与人类对战的结果（天梯图）。具体地，刚刚结束监督学习后的 AlphaStar 可以达到“钻石”级别，而训练到一半（20 天）以及训练完结（40 天）的 AlphaStar 可以达到“大师”级别。这也表明 AlphaStar 已经可以击败绝大多数的普通玩家。

图 13.7(b) 为不同种族间对战的胜率。

图 13.7(c) 为《星际争霸 II》报告的每分钟有效行动分布情况，其中蓝色为 AlphaStar 最终的结果，红色为人类选手的结果，虚线表示平均值。

### 13.5.2 其他实验（消融实验）

AlphaStar 的论文中也使用了消融实验，即控制变量法，来进一步分析每一个约束条件对于对战结果的影响。下面举一个特别的例子。

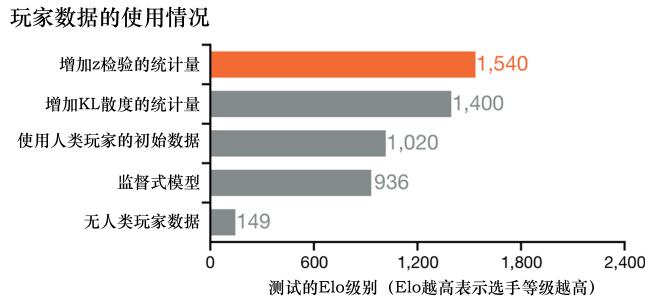


图 13.8 AlphaStar 中人类对局数据使用情况<sup>[1]</sup>

图 13.8 所示为人类对局数据的使用情况。可以看出在没有人类对局数据的情况下，数值仅为 149，但是只要经过了简单的监督学习，对应的数值就可以达到 936，当然使用人类初始化后的强化学习可以达到更好的效果，利用强化学习加监督学习的 KL 散度可以达到接近于完整的利用人类统计量  $Z$  的效果。由此我们可以分析出，AlphaStar 中人类对局数据对于整个模型的表现是很重要的，其并没有完全像 AlphaGo 一样，存在可以不使用人类数据进行训练的情况。

## 13.6 关于 AlphaStar 的总结

关于 AlphaStar 的总结如下。

- (1) AlphaStar 设计了一个高度可融合图像、文本、标量等信息的神经网络架构，并且对于网络设计使用了自回归 (autoregressive) 技巧，从而解耦了结构化的动作空间。
- (2) 其融合了模仿学习和监督学习的内容，例如人类统计量  $Z$  的计算方法。
- (3) 其拥有复杂的深度强化学习方法以及超复杂的训练策略。
- (4) 其完整模型的端到端训练过程需要大量的计算资源。对于此，原文表述如下：每个智能体使用 32 个第三代张量处理单元 (tensor processing unit, TPUs) 进行了 44 天的训练；在训练期间，创建了近 900 个不同的游戏玩家。

## 参考文献

[1] VINYALS O, BABUSCHKIN I, CZARNECKI W M, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning[J]. Nature, 2019, 575(7782): 350-354.

## 附录 A 习题解答

### 第 1 章习题解答

#### 1-1

本质上是智能体与环境的交互。具体地，当智能体在环境中得到当前时刻的状态后，其会基于此状态输出一个动作，这个动作会在环境中被执行并输出下一个状态和当前的这个动作得到的奖励。智能体在环境里存在的目标是最大化期望累积奖励。

#### 1-2

- (1) 强化学习处理的大多是序列数据，其很难像监督学习的样本一样满足独立同分布条件。
- (2) 强化学习有奖励的延迟，即智能体的动作作用在环境中时，环境对于智能体状态的奖励存在延迟，使得反馈不实时。
- (3) 监督学习有正确的标签，模型可以通过标签修正自己的预测来更新模型，而强化学习相当于一个“试错”的过程，其完全根据环境的“反馈”更新对自己最有利的动作。

#### 1-3

- (1) 有试错探索过程，即需要通过探索环境来获取对当前环境的理解。
- (2) 强化学习中的智能体会从环境中获得延迟奖励。
- (3) 强化学习的训练过程中时间非常重要，因为数据都是时间关联的，而不是像监督学习中的数据大部分是满足独立同分布的。
- (4) 强化学习中智能体的动作会影响它从环境中得到的反馈。

#### 1-4

- (1) 算力的提升使我们可以更快地通过试错等方法来使得智能体在环境里面获得更多的信息，从而取得更大的奖励。
- (2) 我们有了深度强化学习这样一个端到端的训练方法，可以把特征提取、价值估计以及决策部分一起优化，这样就可以得到一个更强的决策网络。

#### 1-5

状态是对环境的完整描述，不会隐藏环境信息。观测是对状态的部分描述，可能会遗漏一些信息。在深度强化学习中，我们几乎总是用同一个实值向量、矩阵或者更高阶的张量来表示状态和观测。

#### 1-6

- (1) 策略函数，智能体会用策略函数来选取它下一步的动作，策略包括随机性策略和确定性策略。
- (2) 价值函数，我们用价值函数来对当前状态进行评估，即进入现在的状态可以对后面的奖励带来多大的影响。价值函数的值越大，说明进入该状态越有利。
- (3) 模型，其表示智能体对当前环境状态的理解，它决定系统是如何运行的。

#### 1-7

- (1) 基于价值的智能体。显式学习的是价值函数，隐式地学习智能体的策略。因为这个策略是从学到的价值函数里面推算出来的。
- (2) 基于策略的智能体。其直接学习策略，即直接给智能体一个状态，它就会输出对应动作的概率。当然在基于策略的智能体里面并没有去学习智能体的价值函数。
- (3) 另外还有一种智能体，它把以上两者结合。把基于价值和基于策略的智能体结合起来就有了演员-评论员智能体。这一类智能体通过学习策略函数和价值函数以及两者的交互得到更佳的状态。

#### 1-8

- (1) 基于策略迭代的强化学习方法，智能体会制定一套动作策略，即确定在给定状态下需要采取何种动作，并根据该策略进行操作。强化学习算法直接对策略进行优化，使得制定的策略能够获得最大的奖励；基于价值迭代的强化学习方法，智能体不需要制定显式的策略，它维护一个价值表格或价值函数，并通过这个价值表格或价值函数来选取价值最大的动作。

(2) 基于价值迭代的方法只能应用在离散的环境下，例如围棋或某些游戏领域，对于行为集合规模庞大或是动作连续的场景，如机器人控制领域，其很难学习到较好的结果（此时基于策略迭代的方法能够根据设定的策略来选择连续的动作）。

(3) 基于价值迭代的强化学习算法有 Q-learning、Sarsa 等，基于策略迭代的强化学习算法有策略梯度算法等。

(4) 此外，演员-评论员算法同时使用策略和价值评估来做出决策。其中，智能体会根据策略做出动作，而价值函数会对做出的动作给出价值，这样可以在原有的策略梯度算法的基础上加速学习过程，从而取得更好的效果。

### 1-9

针对是否需要对真实环境建模，强化学习可以分为有模型学习和免模型学习。有模型学习是指根据环境中的经验，构建一个虚拟世界，同时在真实环境和虚拟世界中学习；免模型学习是指不对环境进行建模，直接与真实环境进行交互来学习到最优策略。总体来说，有模型学习相比免模型学习仅仅多出一个步骤，即对真实环境进行建模。免模型学习通常属于数据驱动型方法，需要大量的采样来估计状态、动作及奖励函数，从而优化动作策略。免模型学习的泛化性要优于有模型学习，原因是有模型学习需要对真实环境进行建模，并且虚拟世界与真实环境之间可能还有差异，这限制了有模型学习算法的泛化性。

### 1-10

环境和奖励函数不是我们可以控制的，两者是在开始学习之前就已经事先确定的。我们唯一能做的事情是调整策略，使得智能体可以在环境中得到最大的奖励。另外，策略决定了智能体的行为，策略就是给一个外界的输入，然后它会输出现在应该要执行的动作。

## 第 2 章习题解答

### 2-1

(1) 首先，是有些马尔可夫过程是环状的，它并没有终点，所以我们想避免无穷的奖励。

(2) 另外，我们想把不确定性也表示出来，希望尽可能快地得到奖励，而不是在未来的某个时刻得到奖励。

(3) 接上一点，如果这个奖励是有实际价值的，我们可能更希望立刻就得到奖励，而不是后面才可以得到奖励。

(4) 还有，在有些时候，折扣因子也可以设为 0。当它被设为 0 后，我们就只关注它当前的奖励。我们也可以把它设为 1，设为 1 表示未来获得的奖励与当前获得的奖励是一样的。

所以，折扣因子可以作为强化学习智能体的一个超参数进行调整，然后就会得到不同行为的智能体。

### 2-2

通过矩阵求逆的过程，我们就可以把  $V$  的解析解求出来。但是这个矩阵求逆的过程的复杂度是  $O(N^3)$ ，所以当状态非常多的时候，比如从 10 个状态到 1000 个状态，到 100 万个状态，那么当我们有 100 万个状态的时候，转移矩阵就会是一个 100 万乘 100 万的矩阵。对于这样一个大矩阵进行求逆是非常困难的，所以这种通过解析解去解的方法，只能应用在很小量的马尔可夫奖励过程中。

### 2-3

(1) 蒙特卡洛方法：可用来计算价值函数的值。以本书中的小船示例为例，当得到一个马尔可夫奖励过程后，我们可以从某一个状态开始，把小船放到水中，让它“随波逐流”，这样就会产生一条轨迹，从而得到一个折扣后的奖励  $g$ 。当积累该奖励到一定数量后，直接除以轨迹数量，就会得到其价值函数的值。

(2) 动态规划方法：可用来计算价值函数的值。通过一直迭代对应的贝尔曼方程，最后使其收敛。当最后更新的状态与上一个状态区别不大的时候，通常是小于一个阈值  $\gamma$  时，更新就可以停止。

(3) 以上两者的结合方法：我们也可以使用时序差分学习方法，其为动态规划方法和蒙特卡洛方法的结合。

### 2-4

相对于马尔可夫奖励过程，马尔可夫决策过程多了一个决策过程，其他的定义与马尔可夫奖励过程是类似的。由于多了一个决策，多了一个动作，因此状态转移也多了一个条件，即执行一个动作，导致未来状态的变化，其不仅依赖于当前的状态，也依赖于在当前状态下智能体采取的动作决定的状态变化。对于价值函数，它也多了一个条件，多了一个当前的动作，即当前状态以及采取的动作会决定当前可能得到的奖励的多少。

另外，两者之间是有转换关系的。具体来说，已知一个马尔可夫决策过程以及一个策略  $\pi$  时，我们可以把马尔可夫决策过程转换成马尔可夫奖励过程。在马尔可夫决策过程中，状态的转移函数  $P(s'|s, a)$  是基于它的当前状态和当前动作的，因为我们现在已知策略函数，即在每一个状态，我们知道其采取每一个动作的概率，所以我们可以直接把这个动作进行加和，就可以得到对于马尔可夫奖励过程的一个转移概率。同样地，对于奖励，我们可以把动作去掉，这样就会得到一个类似于马尔可夫奖励过程的奖励。

### 2-5

对于马尔可夫链，它的转移概率是直接决定的，即从当前时刻的状态通过转移概率得到下一时刻的状态值。但是对于马尔可夫决策过程，其中间多了一层动作的输出，即在当前这个状态，首先要决定采取某一种动作，再通过状态转移函数变化到另外一个状态。所以在当前状态与未来状态转移过程中多了一层决策性，这是马尔可夫决策过程与之前的马尔可夫过程的不同之处。在马尔可夫决策过程中，动作是由智能体决定的，所以多了一个组成部分，智能体会采取动作来决定未来的状态转移。

### 2-6

本质来说，当我们取得最佳价值函数后，我们可以通过对 Q 函数进行最大化，从而得到最佳价值。然后，我们直接对 Q 函数取一个让动作最大化的值，就可以直接得到其最佳策略。具体方法如下，

(1) 穷举法（一般不使用）：假设我们有有限个状态、有限个动作可能性，那么每个状态我们可以采取  $A$  种动作策略，那么总共就是  $|A|^{|S|}$  个可能的策略。我们可以把他们穷举一遍，然后算出每种策略的价值函数，对比一下就可以得到最佳策略。但是这种方法的效率极低。

(2) 策略迭代：一种迭代方法，其由两部分组成，以下两个步骤一直在迭代进行，最终收敛，其过程有些类似于机器学习中的 EM 算法（期望-最大化算法）。第一个步骤是策略评估，即当前我们在优化这个策略  $\pi$ ，在优化过程中通过评估从而得到一个更新的策略；第二个步骤是策略提升，即取得价值函数后，进一步推算出它的 Q 函数，得到它的最大值。

(3) 价值迭代：我们一直迭代贝尔曼最优方程，通过迭代，其能逐渐趋向于最佳策略，这是价值迭代方法的核心。我们为了得到最佳的  $V^*$ ，对于每个状态的  $V^*$  值，直接使用贝尔曼最优方程进行迭代，迭代多次之后它就会收敛到最佳策略及其对应的状态，这里是没有策略函数的。

## 第 3 章习题解答

### 3-1

状态、动作、状态转移概率和奖励，分别对应  $(S, A, P, R)$ ，后面有可能会加上折扣因子构成五元组。

### 3-2

可以将强化学习的“学习”流程类比于人类的学习流程。人类学习就是尝试每一条路，并记录尝试每一条路后的最终结果。在人类尝试的过程中，其实就可以慢慢地了解到哪一条路（对应于强化学习中的状态概念）会更好。我们用价值函数  $V(s)$  来定量表达该状态的优劣，然后用 Q 函数来判断在什么状态下做什么动作能够得到最大奖励，在强化学习中我们用 Q 函数来表示状态-动作值。

### 3-3

对于环境和智能体。两者每交互一次以后，智能体都会向环境输出动作，接着环境会反馈给智能体当前时刻的状态和奖励。那么智能体此时会进行两步操作：

- (1) 使用已经训练好的 Q 表格，对应环境反馈的状态和奖励选取对应的动作进行输出。
- (2) 我们已经拥有了  $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$  这几个值，并直接使用  $a_{t+1}$  更新我们的 Q 表格。

### 3-4

Sarsa 算法是 Q 学习算法的改进（这句话可参考论文“On-Line Q-Learning Using Connectionist Systems”的摘要部分），详细描述如下。

(1) 首先，Q 学习是异策略的时序差分学习方法，而 Sarsa 算法是同策略的时序差分学习方法。

(2) 其次，Sarsa 算法在更新 Q 表格的时候所用到的  $a'$  是获取下一个 Q 值时一定会执行的动作。这个动作有可能是用  $\varepsilon$ -贪心方法采样出来的，也有可能是  $\max_Q$  对应的动作，甚至是随机动作。

(3) 但是 Q 学习在更新 Q 表格的时候所用到的 Q 值  $Q(S', a')$  对应的动作不一定是下一步会执行的动作，因为下一步实际会执行的动作可能是因为进一步的探索而得到的。Q 学习默认的动作不是通过行为策略来选取的，它默认  $a'$  为最佳策略对应的动作，所以 Q 学习算法在更新的时候，不需要传入  $a'$ ，即  $a_{t+1}$ 。

(4) 更新公式的对比（区别只在目标计算部分）。

Sarsa 算法的公式： $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$ 。

Q 学习算法的公式： $r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$ 。

总结起来，Sarsa 算法实际上是用固有的策略产生  $S, A, R, S', A'$  这一条轨迹，然后使用  $Q(s_{t+1}, a_{t+1})$  更新原本的 Q 值  $Q(s_t, a_t)$ 。但是 Q 学习算法并不需要知道实际上选择的动作，它默认下一个动作就是 Q 值最大的那个动作。所以 Sarsa 算法的动作通常会更加“保守胆小”，而对应的 Q 学习算法的动作会更加“莽撞激进”。

### 3-5

Sarsa 算法就是一个典型的同策略算法，它只用一个  $\pi$ ，为了兼顾探索和开发，它在训练的时候会显得有点儿“胆小怕事”。它在解决悬崖寻路问题的时候，会尽可能地远离悬崖边，确保哪怕自己不小心向未知区域探索了一些，也还是处在安全区域内，不至于掉入悬崖中。

Q 学习算法是一个比较典型的异策略算法，它有目标策略 (target policy)，用  $\pi$  来表示。此外还有行为策略 (behavior policy)，用  $\mu$  来表示。它分离了目标策略与行为策略，使得其可以大胆地用行为策略探索得到的经验轨迹来优化目标策略。这样智能体就更有可能探索到最优的策略。

比较 Q 学习算法和 Sarsa 算法的更新公式可以发现，Sarsa 算法并没有选取最大值的操作。因此，Q 学习算法是非常激进的，其希望每一步都获得最大的奖励；Sarsa 算法则相对来说偏保守，会选择一条相对安全的迭代路线。

## 第 4 章习题解答

### 4-1

演员做的事情就是操控游戏的摇杆，比如向左、向右、开火等操作；环境就是游戏的主机，负责控制游戏的画面、控制怪物如何移动等；奖励函数就是当执行什么动作、发生什么状况的时候，我们可以得到多少分数，比如击杀一只怪兽得到 20 分、被对手暴击扣除 10 分、完成任务得到 10 分等。

### 4-2

(1) 一部分是环境的行为，即环境的函数内部的参数或内部的规则是什么形式的。 $p(s_{t+1}|s_t, a_t)$  这一项代表的是环境，环境这一项通常是无法控制的，因为它是已经客观存在的，或者其形式是提前制定好的。

(2) 另一部分是智能体的行为，我们能控制的是  $p_\theta(a_t|s_t)$ ，即给定一个状态  $s_t$ ，演员要采取什么样的动作  $a_t$  取决于演员的参数  $\theta$ ，所以这部分是我们可以控制的。随着演员动作的不同，每个同样的轨迹，它会因为不同的概率从而表现出不同的行为。

### 4-3

应该使用梯度上升法，因为要让期望奖励越大越好，所以是梯度上升法。梯度上升法在更新参数的时候要添加梯度信息。要进行梯度上升，我们先要计算期望奖励  $\bar{R}$  的梯度。我们对  $\bar{R}$  取一个梯度，这里只有  $p_\theta(\tau)$  是与  $\theta$  有关的，所以  $p_\theta(\tau)$  为梯度的部分。

### 4-4

策略梯度的公式如下：

$$\begin{aligned} E_{\tau \sim p_\theta(\tau)} [R(\tau) \nabla \log p_\theta(\tau)] &\approx \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log p_\theta(\tau^n) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n | s_t^n) \end{aligned}$$

$p_\theta(\tau)$  里面有两项,  $p(s_{t+1}|s_t, a_t)$  来自环境,  $p_\theta(a_t|s_t)$  来自智能体。 $p(s_{t+1}|s_t, a_t)$  由环境决定, 从而与  $\theta$  无关, 因此  $\nabla \log p(s_{t+1}|s_t, a_t) = 0$ ,  $\nabla p_\theta(\tau) = \nabla \log p_\theta(a_t^n|s_t^n)$ 。

具体来说:

(1) 假设在状态  $s_t$  时执行动作  $a_t$ , 最后发现轨迹  $\tau$  的奖励是正的, 那我们就要增大这一项的概率, 即增大在状态  $s_t$  时执行动作  $a_t$  的概率;

(2) 反之, 在状态  $s_t$  时执行动作  $a_t$  会导致轨迹  $\tau$  的奖励变成负的, 我们就要减小这一项的概率。

#### 4-5

用梯度提升来更新参数, 对于原来的参数  $\theta$ , 可以将原始的  $\theta$  加上更新梯度, 再乘一个学习率。通常学习率也需要调整, 与神经网络一样, 我们可以使用 Adam、RMSProp、SGD 等优化器对其进行调整。

#### 4-6

(1) 增加基线: 为了防止所有奖励都为正, 从而导致每一个状态和动作的变换, 都会使得每一项变换的概率上升, 我们把奖励减去一项  $b$ , 称之为基线。当减去  $b$  后, 就可以让奖励  $R(\tau^n) - b$  有正有负。所以如果得到的总奖励  $R(\tau^n)$  大于  $b$ , 就让它的概率增大。如果总奖励小于  $b$ , 就算它是正的, 值很小也是不好的, 就需要让这一项的概率减小。如果奖励  $R(\tau^n)$  小于  $b$ , 就要让采取这个动作的奖励下降, 这样也符合常理。但是使用基线会让本来奖励很大的“动作”的奖励变小, 从而降低更新速率。

(2) 指派合适的分数: 首先, 原始权重是整个回合的总奖励。现在改成从某个时间点  $t$  开始, 假设动作是在时间点  $t$  被执行的, 从时间点  $t$ , 一直到游戏结束所有奖励的总和大小, 才真正代表这个动作是好的还是不好的; 接下来我们再进一步, 把未来的奖励打一个折扣, 我们称由此得到的奖励的和为折扣回报。

(3) 综合以上两种技巧, 我们将其统称为优势函数, 用  $A$  来代表优势函数。优势函数取决于状态和动作, 即我们需计算的是在某一个状态  $s$  采取某一个动作  $a$  的时候, 优势函数有多大。

(4) 优势函数的意义在于衡量假设我们在某一个状态  $s_t$  执行某一个动作  $a_t$ , 相较于其他可能动作的优势。它在意的不是绝对的好, 而是相对的好, 即相对优势, 因为会减去一个基线  $b$ 。 $A_\theta(s_t, a_t)$  通常可以由一个网络预估出来, 这个网络叫作评论员。

#### 4-7

(1) 两者的更新频率不同。蒙特卡洛强化学习方法是每一个回合更新一次, 即需要经历完整的状态序列后再更新, 比如贪吃蛇游戏, 贪吃蛇“死了”即游戏结束后再更新。而时序差分强化学习方法是每一步就更新一次, 比如贪吃蛇游戏, 贪吃蛇每移动一次(或几次)就进行更新。相对来说, 时序差分强化学习方法比蒙特卡洛强化学习方法更新的频率更高。

(2) 时序差分强化学习方法能够在知道一个小步后就进行学习, 相比于蒙特卡洛强化学习方法, 其更加快速和灵活。

(3) 具体例如: 假如我们要优化开车去公司的通勤时间。对于此问题, 每一次通勤, 我们将到达不同的路口。对于时序差分强化学习方法, 其会对每一个经过的路口计算时间, 例如在路口 A 就开始更新预计到达路口 B、路口 C ……, 以及到达公司的时间; 对于蒙特卡洛强化学习方法, 其不会每经过一个路口就更新时间, 而是到达最终的目的地后, 再修改到达每一个路口和到达公司对应的时间。

#### 4-8

首先我们需要根据一个确定好的策略模型来输出每一个可能动作的概率, 对于所有动作的概率, 我们使用采样方法(或者是随机的方法)选择一个动作与环境进行交互, 同时环境会给我们反馈整个回合的数据。将此回合数据输入学习函数中, 并根据回合数据进行损失函数的构造, 通过 Adam 等优化器的优化, 再更新我们的策略模型。

## 第 5 章习题解答

### 5-1

经典策略梯度的大部分时间花在数据采样上，即当我们的智能体与环境交互后，我们就要进行策略模型的更新。但是对于一个回合我们仅能更新策略模型一次，更新完后我们就要花时间重新采样数据，然后才能再次进行如上的更新。

所以我们可以使用异策略的方法，即使使用另一个不同的策略和演员，与环境进行交互并用所采样的数据进行原先策略的更新。这样等价于使用同一组数据，在同一个回合，我们对整个策略模型更新了多次，这样会更加有效率。

### 5-2

我们可以在重要性采样中将  $p$  替换为任意的  $q$ ，但是本质上要求两者的分布不能差太多，即使我们补偿了不同数据分布的权重  $\frac{p(x)}{q(x)}$ 。 $E_{x \sim p}[f(x)] = E_{x \sim q} \left[ f(x) \frac{p(x)}{q(x)} \right]$ ，当我们对于两者的采样次数都比较多时，最终的结果会是较为接近的。但是通常我们不会取理想数量的采样数据，所以如果两者的分布相差较大，最后结果的方差将会很大。

### 5-3

使用基于异策略的重要性采样后，我们不用  $\theta$  与环境交互，而是由另外一个策略  $\theta'$  进行示范。 $\theta'$  的任务就是示范给  $\theta$  看，它和环境交互，告诉  $\theta$  它与环境交互会发生什么事，以此来训练  $\theta$ 。我们要训练的是  $\theta$ ， $\theta'$  只负责做示范，负责与环境交互，所以采样出来的数据与  $\theta$  本身是没有关系的。所以就可以让  $\theta'$  与环境交互采样大量数据， $\theta$  可以更新参数多次。一直到  $\theta$  训练到一定的程度、参数更新多次以后， $\theta'$  再重新采样，这就是同策略换成异策略的妙处。

### 5-4

本质来说，KL 散度是一个函数，其度量的是两个动作（对应的参数分别为  $\theta$  和  $\theta'$ ）间的行为距离，而不是参数距离。这里的行为距离可以理解为在相同状态下输出动作的差距（概率分布上的差距），概率分布即 KL 散度。

## 第 6 章习题解答

### 6-1

首先深度 Q 网络为基于深度学习的 Q 学习算法，而在 Q 学习中，我们使用表格来存储每一个状态下动作的奖励，即我们在正文中介绍的动作价值函数  $Q(s, a)$ 。但是在我们的实际任务中，状态量通常数量巨大，并且在连续任务中会遇到维度灾难等问题，使用真正的价值函数通常是不切实际的，所以使用了与价值函数近似的表示方法。

### 6-2

与状态和演员直接相关。我们在讨论输出时通常是针对一个演员衡量一个状态的好坏，也就是状态、价值从本质上来说是依赖于演员的。不同的演员在相同的状态下也会有不同的输出。

### 6-3

(1) 基于蒙特卡洛的方法：本质上就是让演员与环境交互。评论员根据统计结果，将演员和状态对应起来，即如果演员看到某一状态  $s_a$ ，将预测接下来的累积奖励有多大，如果看到另一个状态  $s_b$ ，将预测接下来的累积奖励有多大。但是其普适性不好，其需要匹配到所有的状态。如果我们面对的是一个简单的例如贪吃蛇游戏等状态有限的问题还可以应对，但是如果面对的是一个图片型的任务，我们几乎不可能将所有的状态（对应每一帧的图像）都“记录”下来。总之，其不能对未出现过的输入状态进行对应价值的输出。

(2) 基于蒙特卡洛的网络方法：为了弥补上面描述的基于蒙特卡洛的方法的不足，我们将其中的状态价值函数  $V_\pi(s)$  定义为一个网络，其可以对于从未出现过的输入状态，根据网络的泛化和拟合能力，“估测”出一个价值输出。

(3) 基于时序差分的网络方法，即基于时序差分的网络：与我们在前 4 章介绍的蒙特卡洛方法与时序差分方法的区别一样，基于时序差分的网络方法和基于蒙特卡洛的网络方法的区别也相同。在基于蒙特卡洛的方法中，每次我们都要计算累积奖励，也就是从某一个状态  $s_a$  一直到游戏结束的时候，得到的所有奖励的总和。所以要应用基于蒙特卡洛的方法时，我们必须至少把游戏玩到结束。但有些游戏要玩到游戏结束才能够更新网络花费的时间太长了，因此我们会采用基于时序差分的网络方法。基于时序差分的网络方法不需要把游戏玩到结束，只要在游戏某一个状态  $s_t$  的时候，采取动作  $a_t$  得到奖励  $r_t$ ，进入状态  $s_{t+1}$ ，就可以应用基于时序差分的网络方法。其公式与之前介绍的时序差分方法类似，即  $V_\pi(s_t) = V_\pi(s_{t+1}) + r_t$ 。

(4) 基于蒙特卡洛方法和基于时序差分方法的区别在于：蒙特卡洛方法本身具有很大的随机性，我们可以将其  $G_a$  视为一个随机变量，所以其最终的偏差很大。而对于时序差分，其具有随机的变量  $r$ 。因为在状态  $s_t$  采取同一个动作，所得的奖励也不一定是一样的，所以对于时序差分方法来说， $r$  是一个随机变量。但是相对于蒙特卡洛方法的  $G_a$  来说， $r$  的随机性非常小，这是因为  $G_a$  本身就是由很多的  $r$  组合而成的。从另一个角度来说，在时序差分方法中，我们的前提是  $r_t = V_\pi(s_{t+1}) - V_\pi(s_t)$ ，但是我们通常无法保证  $V_\pi(s_{t+1})$ 、 $V_\pi(s_t)$  计算的误差为 0。所以当  $V_\pi(s_{t+1})$ 、 $V_\pi(s_t)$  计算得不准确，得到的结果也会是不准确的。总之，两者各有优劣。

(5) 目前，基于时序差分的方法是比较常用的，基于蒙特卡洛的方法其实是比较少用的。

#### 6-4

理想状态下，我们期望对于一个输入状态，输出其无误差的奖励价值。对于价值函数，如果输入状态是  $s_a$ ，正确的输出价值应该是  $G_a$ 。如果输入状态是  $s_b$ ，正确的输出价值应该是  $G_b$ 。所以在训练的时候，其就是一个典型的机器学习中的回归问题。我们实际中需要输出的仅仅是一个非精确值，即我们希望在输入状态  $s_a$  的时候，输出价值与  $G_a$  越近越好；输入  $s_b$  的时候，输出价值与  $G_b$  越近越好。其训练方法与我们在训练卷积神经网络等深度神经网络时的方法类似。

#### 6-5

基于时序差分网络的核心函数为  $V_\pi(s_t) = V_\pi(s_{t+1}) + r_t$ 。我们将状态  $s_t$  输入网络，因为将  $s_t$  输入网络会得到输出  $V_\pi(s_t)$ ，同样将  $s_{t+1}$  输入网络会得到  $V_\pi(s_{t+1})$ 。同时核心函数  $V_\pi(s_t) = V_\pi(s_{t+1}) + r_t$  告诉我们， $V_\pi(s_t)$  减  $V_\pi(s_{t+1})$  的值应该是  $r_t$ 。我们希望它们两个相减的损失值与  $r_t$  尽可能地接近。这也是网络的优化目标，我们称之为损失函数。

#### 6-6

(1) 状态价值函数的输入是一个状态，它根据状态计算出当前这个状态以后的累积奖励的期望值是多少。

(2) 动作价值函数的输入是状态-动作对，即在某一个状态采取某一个动作，同时假设我们都使用策略  $\pi$ ，得到的累积奖励的期望值是多少。

#### 6-7

(1) 使用状态-动作对表示时，即当 Q 函数的输入是状态-动作对时，输出就是一个标量。

(2) 仅使用状态表示时，即当 Q 函数的输入仅是一个状态时，输出就是多个价值。

#### 6-8

首先， $\pi'(s) = \arg \max_a Q_\pi(s, a)$  计算而得，其表示假设我们已经学习出  $\pi$  的 Q 函数，对于某一个状态  $s$ ，把所有可能的动作  $a$  一一代入这个 Q 函数，看看哪一个动作  $a$  可以让 Q 函数的价值最大，那么该动作就是  $\pi'$  将会执行的动作。所以根据以上方法决定动作的策略  $\pi'$  一定比原来的策略  $\pi$  要好，即  $V_{\pi'}(s) \geq V_\pi(s)$ 。

#### 6-9

(1)  $\varepsilon$ -贪心：我们有  $1 - \varepsilon$  的概率（通常  $\varepsilon$  很小）完全按照 Q 函数决定动作，但是有  $\varepsilon$  的概率使得动作是随机的。通常在实现上， $\varepsilon$  的值会随着时间递减。也就是在最开始的时候，因为还不知道哪个动作是比较好的，所以我们会花比较大的力气做探索。接下来随着训练的次数越来越多，我们已经比较确定哪一种策略是比较好的，就会减少探索，从而把  $\varepsilon$  的值变小，主要根据 Q 函数来决定未来的动作，随机性就

会变小。

(2) 玻尔兹曼探索：这个方法比较像策略梯度。在策略梯度里面，网络的输出是一个期望动作空间上的一个概率分布，我们根据概率分布去采样。所以也可以根据 Q 值确定一个概率分布，假设某一个动作的 Q 值越大，代表它越好，我们采取这个动作的概率就越高。

### 6-10

(1) 首先，在强化学习的整个过程中，最花时间的过程是与环境交互，使用 GPU 乃至 TPU 来训练网络相对来说是比较快的。而用回放缓冲区可以减少与环境交互的次数。因为在训练的时候，我们的经验不需要通通来自于某一个策略（或者当前时刻的策略）。一些由过去的策略所得到的经验可以放在回放缓冲区中被使用多次，被反复地再利用，这样采样到的经验才能被高效地利用。

(2) 另外，在训练网络的时候，我们其实希望一个批量里面的数据越多样越好。如果一个批量里面的数据都是同性质的，我们训练出的模型的拟合能力可能不会很乐观。如果一个批量里面都是一样的数据，在训练的时候，拟合效果会比较差。如果回放缓冲区里面的经验通通来自于不同的策略，那么采样到的一个批量里面的数据会是比较多样化的。这样可以保证我们的模型的性能至少不会很差。

### 6-11

没影响。这并不是因为过去的  $\pi$  与现在的  $\pi'$  很相似，就算过去的  $\pi$  不是很相似，其实也是没有关系的。主要的原因是我们并不是去采样一条轨迹，我们只能采样一个经验，所以与是不是异策略是没有关系的。就算是异策略，就算是这些经验不是来自  $\pi$ ，我们还是可以使用这些经验来估测  $Q_\pi(s, a)$ 。

## 第 7 章习题解答

### 7-1

因为实际应用时，需要让  $Q(s_t, a_t)$  与  $r_t + \max_a Q(s_{t+1}, a)$  尽可能相等，即与我们的目标越接近越好。可以发现，目标值很容易一不小心就被设置得太高，因为在计算该目标值的时候，我们实际上在做的事情是看哪一个动作  $a$  可以得到最大的 Q 值，就把它加上去，使其成为我们的目标。

例如，现在有 4 个动作，本来它们得到的 Q 值都是差不多的，它们得到的奖励也都是差不多的，但是在估算的时候是有误差的。如果第 1 个动作被高估了，那目标就会执行该动作，然后就会选这个高估的动作的 Q 值加上  $r_t$  当作目标值。如果第 4 个动作被高估了，那目标就会选第 4 个动作的 Q 值加上  $r_t$  当作目标值。所以目标总是会选那个 Q 值被高估的动作，我们也总是会选那个奖励被高估的动作的 Q 值当作 Q 值的最大值的结果去加上  $r_t$  当作新目标值，因此目标值总是太大。

### 7-2

我们可以使用双深度 Q 网络解决这个问题。首先，在双深度 Q 网络里面，选动作的 Q 函数与计算价值的 Q 函数不同。在深度 Q 网络中，需要穷举所有的动作  $a$ ，把每一个动作  $a$  都代入 Q 函数并计算哪一个动作  $a$  反馈的 Q 值最大，把这个 Q 值加上  $r_t$ 。但是对于双深度 Q 网络的两个 Q 网络，第一个 Q 网络决定哪一个动作的 Q 值最大，以此来决定选取的动作。我们的 Q 值是用  $Q'$  算出来的，这样有什么好处呢？为什么这样就可以避免过度估计的问题呢？假设我们有两个 Q 函数，如果第一个 Q 函数高估了它现在选出来的动作  $a$  的值，那没关系，只要第二个 Q 函数  $Q'$  没有高估这个动作  $a$  的值，计算得到的就还是正常值。假设反过来是  $Q'$  高估了某一个动作的值，那也不会产生过度估计的问题。

### 7-3

在双深度 Q 网络中存在两个 Q 网络，一个是目标的 Q 网络，一个才是真正需要更新的 Q 网络。具体实现方法是使用需要更新的 Q 网络选动作，然后使用目标的 Q 网络计算价值。双深度 Q 网络相较于深度 Q 网络的更改是最少的，它几乎没有增加任何的运算量，甚至连新的网络都不需要。唯一要改变的就是在找最佳动作  $a$  的时候，本来使用  $Q'$  来计算，即用目标的 Q 网络来计算，现在改成用需要更新的 Q 网络来计算。

### 7-4

对于  $Q(s, a)$ ，其对应的状态由于为表格的形式，因此是离散的，而实际中的状态却不是离散的。对

于  $Q(s, a)$  的计算公式—— $Q(s, a) = V(s) + A(s, a)$ 。其中的  $V(s)$  对于不同的状态都有值， $A(s, a)$  对于不同的状态都有不同的动作对应的值。所以从本质上来说，我们最终矩阵  $Q(s, a)$  的结果是将每一个  $V(s)$  加到矩阵  $A(s, a)$  中得到的。从模型的角度考虑，我们的网络直接改变的不是  $Q(s, a)$ ，而是改变的  $V, A$ 。但是有时我们更新时不一定会将  $V(s)$  和  $Q(s, a)$  都更新。将状态和动作对分成两个部分后，我们就不需要将所有的状态-动作对都采样一遍，我们可以使用更高效的估计  $Q$  值的方法将最终的  $Q(s, a)$  计算出来。

### 7-5

**优势：**时序差分方法只采样了一步，所以某一步得到的数据是真实值，接下来的都是  $Q$  值估测出来的。使用蒙特卡洛和时序差分平衡方法采样比较多步，如采样  $N$  步才估测价值，所以估测的部分所造成的影响就会比较小。

**劣势：**因为智能体的奖励比较多，所以当我们把  $N$  步的奖励加起来时，对应的方差就会比较大。为了缓解方差大的问题，我们可以通过调整  $N$  值，在方差与不精确的  $Q$  值之间取得一个平衡。这里介绍的参数  $N$  是超参数，需要微调参数  $N$ ，例如是要多采样 3 步、还是多采样 5 步。

## 第 8 章习题解答

### 8-1

在深度  $Q$  网络中，只要能够估计出  $Q$  函数，就可以找到一个比较好的策略。同样地，只要能够估计出  $Q$  函数，就可以增强对应的策略。因为估计  $Q$  函数是一个比较容易的回归问题，在这个回归问题中，我们可以时刻观察模型训练的效果是不是越来越好（一般情况下我们只需要关注回归的损失有没有下降，就可以判断模型学习得好不好），所以估计  $Q$  函数相较于学习一个策略来说是比较容易的。只需要估计  $Q$  函数，就可以保证现在一定会得到比较好的策略，同样其也比较容易操作。对比来说，策略梯度方法中的优化目标是最大化总回报，但是我们很难找到一个明确的损失函数来进行优化，其本质上是一个策略搜索问题，也就是一个无约束的优化问题。

### 8-2

我们在日常生活中常见的问题大都是包含连续动作的，例如智能体要进行自动驾驶，其就需要决定方向盘要左转几度或右转几度，这就是连续的动作；假设智能体是一个机器人，它身上有 50 个关节，它的每一个动作就对应到这 50 个关节的角度，这些角度也是连续的。

然而在使用深度  $Q$  网络时，很重要的一步是要求能够解决对应的优化问题。当我们预估出  $Q$  函数  $Q(s, a)$  以后，必须要找到一个动作，它可以让  $Q(s, a)$  最大。假设动作是离散的，那么动作  $a$  的可能性是有限的。但如果动作是连续的，我们就不能像对离散的动作一样，穷举所有可能的动作了。

为了解决这个问题，有以下几种方案。

(1) 第一个方案：我们可以使用采样方法，即随机采样出  $N$  个可能的动作，然后一个一个代入  $Q$  函数中，计算对应的  $N$  个  $Q$  值，并比较哪一个最大。但是这个方案因为使用采样方法所以不会非常精确。

(2) 第二个方案：我们将这个连续动作问题，建模为一个优化问题，从而可以用梯度上升去最大化我们的目标函数。具体地，我们将动作视为变量，使用梯度上升更新动作对应的  $Q$  值。但是这个方案通常时间花销比较大，因为其需要迭代计算。

(3) 第三个方案：设计一个特别的网络架构，即设计一个特别的  $Q$  函数，使得求解让  $Q$  函数最大化的动作  $a$  变得非常容易。也就是这里的  $Q$  函数不是一个广义的  $Q$  函数，我们可以使用特殊方法设计  $Q$  函数，使得寻找让这个  $Q$  函数最大的动作  $a$  非常容易。但是这个方案的  $Q$  函数不能随意设计，其必须有一些额外的限制。

(4) 第四个方案：不用深度  $Q$  网络，毕竟用其处理连续动作比较麻烦。

## 第 9 章习题解答

### 9-1

在传统的方法中，我们有一个策略  $\pi$  以及一个初始的演员与环境交互、收集数据以及反馈。通过每一步得到的反馈，我们进一步更新我们的策略  $\pi$ ，通常我们使用的更新方式是策略梯度。但是对于演员-评论员算法，我们不是直接使用每一步得到的数据和反馈进行策略  $\pi$  的更新，而是使用这些数据和反馈进行价值函数的估计，这里我们通常使用的算法包括时序差分和蒙特卡洛等算法以及基于它们的优化算法。接下来我们再基于价值函数来更新策略，公式如下：

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} (r_t^n + V_\pi(s_{t+1}^n) - V_\pi(s_t^n)) \nabla \log p_\theta(a_t^n | s_t^n)$$

其中  $r_t^n + V_\pi(s_{t+1}^n) - V_\pi(s_t^n)$  为优势函数。我们通过以上方法得到新的策略后，再与环境交互，然后重复预估价值函数的操作，用价值函数来更新我们的策略。以上的整个方法我们称为优势演员-评论员算法。

### 9-2

(1) 预估两个网络：一个是价值网络；另外一个是策略网络。价值网络的输入是一个状态，输出是一个标签；策略网络的输入是一个状态，输出是一个动作的分布。这两个网络中，演员和评论员的输入都是状态，所以它们前面几层是可以共享的。例如，玩雅达利游戏时，输入都是图片。输入的图片都非常复杂，且比较大，通常前期我们都会用一些卷积神经网络来处理这些图片，把图片抽象成深层次的特征，这些网络对演员与评论员网络来说是可以共用的。我们可以让演员与评论员的前面几层共用同一组参数，这一组参数可能是卷积神经网络中的参数。先把输入的像素变成比较高维度的特征信息，然后输入演员网络决定要采取什么样的动作，评论员网络使用价值函数计算期望奖励。

(2) 探索机制：其目的是对策略  $\pi$  的输出分布进行限制，从而使得分布的熵不要太小，即希望不同的动作被采用的概率平均一些。这样在测试的时候，智能体才会多尝试各种不同的动作，才会对环境进行充分探索，从而得到比较好的结果。

### 9-3

异步优势演员-评论员算法，即算法一开始会有一个全局网络，其包含策略部分和价值部分。假设它的参数是  $\theta_1$ ，假设对于每一个演员都用一个 CPU 训练，每一个演员工作前都会将全局网络的参数复制进来。然后演员与环境进行交互，每一个演员与环境交互后，都会计算出梯度并且更新全局网络的参数。这里要注意的是，所有的演员都是并行运行的。所以每个演员都是在全局网络复制了参数以后，执行完再把参数传回去。所以当第一个演员执行完想要把参数传回去的时候，本来它要的参数是  $\theta_1$ ，等它把梯度传回去的时候，可能原来的参数已经被覆盖，变成  $\theta_2$  了。

### 9-4

(1) 把  $Q(s, a)$  换成了  $\pi$ 。经典的 Q 学习算法是用  $Q(s, a)$  来决定在状态  $s_t$  产生哪一个动作  $a_t$ ，路径衍生策略梯度是直接用  $\pi$  来决定。面对前者，我们需要解决最大值的问题，现在的路径衍生策略梯度直接训练了一个演员网络。其输入状态  $s_t$  就会告诉我们应该采取哪一个动作  $a_t$ 。综上，经典的 Q 学习算法输入状态  $s_t$ ，采取哪一个动作  $a_t$  是  $Q(s, a)$  决定的，在路径衍生策略梯度里面，我们会直接用  $\pi$  来决定。

(2) 经典的 Q 学习算法计算在  $s_{i+1}$  下对应的策略采取的动作  $a$  得到的 Q 值，我们会采取让  $\hat{Q}$  最大的动作  $a$ 。现在的路径衍生策略梯度因为我们不需要再求解决最大化的问题，所以我们直接把状态  $s_{i+1}$  代入策略  $\pi$  中，就会得到在状态  $s_{i+1}$  下，哪一个动作会带给我们最大的 Q 值，就执行这个动作。在 Q 函数中，有两个 Q 网络，一个是真正的 Q 网络，另外一个是目标 Q 网络。实际上在执行时，也会有两个演员网络，一个真正要学习的演员网络  $\pi$  和一个目标演员网络  $\hat{\pi}$ 。

(3) 经典的 Q 学习算法只需要学习 Q 函数，路径衍生策略梯度需要多学习一个策略  $\pi$ ，其目的在于最大化 Q 函数，希望得到的演员可以让 Q 函数的输出尽可能的大，这与生成对抗网络里面的生成器的概念类似。

(4) 与原来的 Q 函数一样，我们要把目标 Q 网络取代掉，路径衍生策略梯度中也要把目标策略取代掉。

## 第 10 章习题解答

### 10-1

设计奖励、好奇心驱动的奖励、课程学习、逆课程学习、分层强化学习等。

### 10-2

主要的问题是我们人为设计的奖励需要领域知识，需要我们自己设计出让环境与智能体更好地交互的奖励，这需要不少的经验知识，并且需要我们根据实际的效果进行调整。

### 10-3

内在好奇心模块代表好奇心驱动技术中增加新的奖励函数以后的奖励函数。具体来说，其在更新计算时会考虑 3 个新的部分，分别是状态  $s_1$ 、动作  $a_1$  和状态  $s_2$ 。根据  $s_1$ 、 $a_1$ 、 $a_2$ ，它会输出另外一个新的奖励  $r_1^i$ 。所以在内在好奇心模块中，我们的总奖励并不是只有  $r$  而已，还有  $r^i$ 。它不是只把所有的  $r$  相加，还把所有  $r^i$  相加一并当作总奖励。所以，基于内在好奇心模块的智能体在与环境交互的时候，不只是希望  $r$  越大越好，还同时希望  $r^i$  越大越好，希望从内在好奇心模块里面得到的总奖励越大越好。

对于如何设计内在好奇心模块，其输入就像前面所说的一样，包括 3 部分，即现在的状态  $s_1$ 、在这个状态采取的动作  $a_1$ 、下一个状态  $s_{t+1}$ ，对应的输出就是奖励  $r_1^i$ 。输入、输出的映射是通过网络构建的，其使用状态  $s_1$  和动作  $a_1$  去预测下一个状态  $\hat{s}_{t+1}$ ，然后继续评判预测的状态  $\hat{s}_{t+1}$  和真实状态  $s_{t+1}$  的相似性，越不相似得到的奖励就越大。通俗来说这个奖励就是，如果未来的状态越难被预测，那么得到的奖励就越大。这就是好奇心机制，其倾向于让智能体做一些风险比较大的动作，从而提高其探索的能力。

同时，为了进一步增强网络的表达能力，我们通常将内在好奇心模块的输入优化为特征提取，特征提取器的输入就是状态，输出是一个特征向量，其可以表示这个状态最主要和最重要的特征，把没有意义的事物过滤。

## 第 11 章习题解答

### 11-1

行为克隆、逆强化学习或者称为逆最优控制。

### 11-2

(1) 首先，如果只收集专家的示范（看到某一个状态输出的动作），那么所有的结果会是非常有限的。所以我们要收集专家在各种极端状态下的动作或者说要收集更多、更复杂的数据，可以使用数据集聚合方法。

(2) 另外，使用传统意义上的行为克隆，智能体会完全复制专家的行为，不管专家的行为是否合理，智能体都会硬把它记下来。智能体是一个网络，网络的容量是有限的。就算给网络足够的训练数据，它在训练数据集上得到的正确率往往也不是 100%。所以这个时候，什么该学、什么不该学就变得很重要。实际上，极少数专家的行为是没有意义的，但是使用它们的示范至少不会产生较坏的影响。

(3) 还有，在进行行为克隆的时候，训练数据和测试数据往往是不匹配的。我们可以用数据集聚合来缓解这个问题。具体来说，在训练和测试的时候，数据分布是不同的。因为在强化学习中，动作会影响到接下来的状态。我们先有状态  $s_1$ ，然后采取动作  $a_1$ ，动作  $a_1$  会决定接下来的状态  $s_2$ 。如果  $\pi^*$  与  $\hat{\pi}$  一模一样，那么我们训练时看到的状态与测试时看到的状态会是一样的，这样模型的泛化性能就会变得比较差。而且， $\pi^*$  和  $\hat{\pi}$  可能有一点儿误差，虽然这个误差在监督学习中，由于每一个样本都是独立的，因此影响不大，但对强化学习来说，可能在某个地方，也许智能体无法完全复制专家的行为，最后得到的结果就会差很多。所以行为克隆并不能够完全解决模仿学习的问题，我们可以使用另外一个比较好的方法，即逆强化学习。

### 11-3

首先，我们有一个专家，其策略为  $\hat{\pi}$ ，这个专家负责与环境交互，给我们  $\hat{\tau}_1 \sim \hat{\tau}_n$ ，我们需要将其中的状态-动作序列都记录下来。然后对于演员，其策略为  $\pi$ ，也需要进行一样的交互和序列的记录。接着我们需要指定一个奖励函数，并且保证专家对应的分数一定要比演员的要高，用这个奖励函数继续学习并更新

我们的训练，同时套用一般条件下的强化学习方法进行演员网络的更新。在这个过程中，我们也要同时进行一开始指定的奖励函数的更新，使得演员得分越来越高，但是不超过专家的得分。最终的奖励函数应该让专家和演员对应的奖励函数都达到比较高的分数，并且从最终的奖励函数中无法分辨出两者。

#### 11-4

在生成对抗网络中，我们有一些比较好的图片数据集，也有一个生成器，一开始其不知道要生成什么样的图片，只能随机生成。另外，我们有一个判别器，其用来给生成的图片打分，专家生成的图片得分高，生成器生成的图片得分低。有了判别器以后，生成器会想办法去“骗”判别器。生成器希望判别器也给它生成的图片打高分。整个过程与逆强化学习的过程是类似的。我们一一对应起来看。

(1) 生成的图片就是专家的判别结果，生成器就是演员，生成器会生成很多的图片并让演员与环境进行交互，从而产生很多轨迹。这些轨迹与环境交互的记录等价于生成对抗网络中的生成图片。

(2) 逆强化学习中的奖励函数就是判别器。奖励函数给专家的实例打高分，给演员的交互结果打低分。

(3) 考虑两者的过程，在逆强化学习中，演员会想办法从已经学到的奖励函数中获得高分，然后迭代地循环。这个过程其实是与生成对抗网络的训练过程一致的。

## 第 12 章习题解答

### 12-1

(1) 对于随机性策略  $\pi_\theta(a_t|s_t)$ ，我们输入某一个状态  $s$ ，采取某一个动作  $a$  的可能性并不是百分之百的，而是有一个概率的，就好像抽奖一样，根据概率随机抽取一个动作。

(2) 对于确定性策略  $\mu_\theta(s_t)$ ，其没有概率的影响。当神经网络的参数固定之后，输入同样的状态，必然输出同样的动作，这就是确定性策略。

### 12-2

首先需要说明的是，对于连续动作的控制空间，Q 学习、深度 Q 网络等算法是没有办法处理的，所以我们需要使用神经网络进行处理，因为其可以既输出概率值，也可以输出确定的策略  $\mu_\theta(s_t)$ 。

(1) 要输出离散动作，最后输出的激活函数使用 Softmax 即可。其可以保证输出的是动作概率，而且所有的动作概率加和为 1。

(2) 要输出连续的动作，可以在输出层中加一层 tanh 激活函数，其可以把输出限制到  $[-1, 1]$ 。我们得到这个输出后，就可以根据实际动作的一个范围再做缩放，然后将其输出给环境。比如神经网络输出一个浮点数 2.8，经过 tanh 激活函数之后，它就可以被限制在  $[-1, 1]$ ，输出 0.99。假设小车的速度的动作范围是  $[-2, 2]$ ，那我们就按比例将之从  $[-1, 1]$  扩大到  $[-2, 2]$ ，0.99 乘 2，最终输出的就是 1.98，将其作为小车的速度或者推小车的力输出给环境。

## 附录 B 面试题解答

### 第 1 章面试题解答

#### 1-1

强化学习包含环境、动作和奖励 3 部分，其本质是智能体通过与环境的交互，使其做出的动作对应的决策得到的总奖励最大，或者说是期望最大。

#### 1-2

首先强化学习和无监督学习是不需要有标签样本的，而监督学习需要许多有标签样本来进行模型的构建和训练。其次对于强化学习与无监督学习，无监督学习直接基于给定的数据进行建模，寻找数据或特征中隐藏的结构，一般对应聚类问题；强化学习需要通过延迟奖励学习策略来得到模型与目标的距离，这个距离可以通过奖励函数进行定量判断，这里我们可以将奖励函数视为正确目标的一个稀疏、延迟形式。另外，强化学习处理的多是序列数据，样本之间通常具有强相关性，但其很难像监督学习的样本一样满足独立同分布条件。

**1-3**

7个字总结就是“多序列决策问题”，或者说对应的模型未知，需要通过学习逐渐逼近真实模型的问题。并且当前的动作会影响环境的状态，即具有马尔可夫性的问题。同时应满足所有状态是可重复到达的条件，即满足可学习条件。

**1-4**

深度学习中的损失函数的目的是使预测值和真实值之间的差距尽可能小，而强化学习中的损失函数的目的是使总奖励的期望尽可能大。

**1-5**

我认为两者的区别主要在于是否需要对真实的环境进行建模，免模型方法不需要对环境进行建模，直接与真实环境进行交互即可，所以其通常需要较多的数据或者采样工作来优化策略，这也使其对于真实环境具有更好的泛化性能；而有模型方法需要对环境进行建模，同时在真实环境与虚拟环境中进行学习，如果建模的环境与真实环境的差异较大，那么会限制其泛化性能。现在通常使用有模型方法进行模型的构建工作。

## 第2章面试题解答

**2-1**

马尔可夫过程是一个二元组  $\langle S, P \rangle$ ， $S$  为状态集合， $P$  为状态转移函数；

马尔可夫决策过程是一个五元组  $\langle S, P, A, R, \gamma \rangle$ ，其中  $R$  表示从  $S$  到  $S'$  能够获得的奖励期望， $\gamma$  为折扣因子， $A$  为动作集合；

马尔可夫最重要的性质是下一个状态只与当前状态有关，与之前的状态无关，也就是  $p(s_{t+1}|s_t) = p(s_{t+1}|s_1, s_2, \dots, s_t)$ 。

**2-2**

我们求解马尔可夫决策过程时，可以直接求解贝尔曼方程或动态规划方程：

$$V(s) = R(S) + \gamma \sum_{s' \in S} p(s'|s)V(s')$$

特别地，其矩阵形式为  $V = R + \gamma PV$ 。但是贝尔曼方程很难求解且计算复杂度较高，所以可以使用动态规划、蒙特卡洛以及时序差分等方法求解。

**2-3**

如果不具备马尔可夫性，即下一个状态与之前的状态也有关，若仅用当前的状态来求解决策过程，势必导致决策的泛化能力变差。为了解决这个问题，可以利用循环神经网络对历史信息建模，获得包含历史信息的状态表征，表征过程也可以使用注意力机制等手段，最后在表征状态空间求解马尔可夫决策过程问题。

**2-4**

- (1) 基于状态价值函数的贝尔曼方程:  $V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r(s,a) + \gamma V_\pi(s')]$ ;
- (2) 基于动作价值函数的贝尔曼方程:  $Q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r(s',a) + \gamma V_\pi(s')]$ 。

**2-5**

最佳价值函数的定义为  $V^*(s) = \max_\pi V_\pi(s)$ ，即我们搜索一种策略  $\pi$  来让每个状态的价值最大。 $V^*$  就是到达每一个状态其的最大价值，同时我们得到的策略就可以说是最佳策略，即  $\pi^*(s) = \arg \max_\pi V_\pi(s)$ 。最佳策略使得每个状态的价值函数都取得最大值。所以如果我们可以得到一个最佳价值函数，就可以说某一个马尔可夫决策过程的环境被解。在这种情况下，其最佳价值函数是一致的，即其达到的上限的值是一致的，但这里可能有多个最佳策略对应于相同的最佳价值。

**2-6**

$n$  越大，方差越大，期望偏差越小。价值函数的更新公式如下：

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[ \sum_{i=1}^n \gamma^{i-1} r_{t+i} + \gamma^n \max_a Q(S', a) - Q(S, A) \right]$$

### 第 3 章面试题解答

#### 3-1

同策略和异策略的根本区别在于生成样本的策略和参数更新时的策略是否相同。对于同策略，行为策略和要优化的策略是同一策略，更新了策略后，就用该策略的最新版本对数据进行采样；对于异策略，其使用任意行为策略来对数据进行采样，并利用其更新目标策略。例如，Q 学习在计算下一状态的预期奖励时使用了最大化操作，直接选择最优动作，而当前策略并不一定能选择到最优的动作，因此这里生成样本的策略和学习时的策略不同，所以 Q 学习算法是异策略算法；相对应的 Sarsa 算法则是基于当前的策略直接执行一次动作选择，然后用动作和对应的状态更新当前的策略，因此生成样本的策略和学习时的策略相同，所以 Sarsa 算法为同策略算法。

#### 3-2

Q 学习是通过计算最优动作价值函数来求策略的一种时序差分的学习方法，其更新公式为

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

其是异策略的，由于 Q 更新使用了下一个时刻的最大值，因此其只关心哪个动作使得  $Q(s_{t+1}, a)$  取得最大值，而实际上到底采取了哪个动作（行为策略），Q 学习并不关心。这表明优化策略并没有用到行为策略的数据，所以说它是异策略的。

#### 3-3

Sarsa 算法可以算是 Q 学习算法的改进，其更新公式为

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma Q(s', a') - Q(s, a)]$$

其为同策略的，Sarsa 算法必须执行两次动作得到  $(s, a, r, s', a')$  才可以更新一次；而且  $a'$  是在特定策略  $\pi$  的指导下执行的动作，因此估计出来的  $Q(s, a)$  是在该策略  $\pi$  下的 Q 值，样本生成用的  $\pi$  和估计的  $\pi$  是同一个，因此是同策略。

#### 3-4

(1) 生成策略上的差异，前者确定，后者随机。基于价值的方法中动作-价值对的估计值最终会收敛（通常是不同的数，可以转化为  $0 \sim 1$  的概率），因此通常会获得一个确定的策略；基于策略的方法不会收敛到一个确定的值，另外他们会趋向于生成最佳随机策略。如果最佳策略是确定的，那么最优动作对应的值函数的值将远大于次优动作对应的值函数的值，值函数的大小代表概率的大小。

(2) 动作空间是否连续，前者离散，后者连续。基于价值的方法，对于连续动作空间问题，虽然可以将动作空间离散化处理，但离散间距的选取不易确定。过大的离散间距会导致算法取不到最优动作，会在最优动作附近徘徊；过小的离散间距会使得动作的维度增大，会和高维度动作空间一样导致维度灾难，影响算法的速度。而基于策略的方法适用于连续的动作空间，在连续的动作空间中，可以不用计算每个动作的概率，而是通过正态分布选择动作。

(3) 基于价值的方法，例如 Q 学习算法，是通过求解最优价值函数而间接地求解最优策略；基于策略的方法，例如 REINFORCE 等算法直接将策略参数化，通过策略搜索、策略梯度或者进化方法来更新参数以最大化回报。基于价值的方法不易扩展到连续动作空间，并且当同时采用非线性近似、自举等策略时会有收敛问题。策略梯度具有良好的收敛性。

(4) 另外，对于价值迭代和策略迭代，策略迭代有两个循环，一个是在策略估计的时候，为了求当前策略的价值函数需要迭代很多次；另一个是外面的大循环，即策略评估、策略提升。价值迭代算法则是一步到位，直接估计最优价值函数，因此没有策略提升环节。

**3-5**

时序差分算法是使用广义策略迭代来更新 Q 函数的方法，核心是使用自举，即价值函数的更新使用下一个状态的价值函数来估计当前状态的价值。也就是使用下一步的 Q 值  $Q(s_{t+1}, a_{t+1})$  来更新当前步的 Q 值  $Q(s_t, a_t)$ 。完整的计算公式如下：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})]$$

**3-6**

蒙特卡洛方法是无偏估计，时序差分方法是有偏估计；蒙特卡洛方法的方差较大，时序差分方法的方差较小，原因在于时序差分方法中使用了自举，实现了基于平滑的效果，导致估计的价值函数的方差更小。

**3-7**

相同点：都用于进行价值函数的描述与更新，并且所有方法都基于对未来事件的展望计算一个回溯值。

不同点：蒙特卡洛方法和时序差分方法属于免模型方法，而动态规划属于有模型方法；时序差分方法和蒙特卡洛方法，因为都是免模型的方法，所以对于后续状态的获知也都是基于试验的方法；时序差分方法和动态规划方法的策略评估，都能基于当前状态的下一步预测情况来得到对于当前状态的价值函数的更新。

另外，时序差分方法不需要等到试验结束后才能进行当前状态的价值函数的计算与更新，而蒙特卡洛方法需要与环境交互，产生一整条马尔可夫链并直到最终状态才能进行更新。时序差分方法和动态规划方法的策略评估不同之处为免模型和有模型，动态规划方法可以凭借已知转移概率推断出后续的状态情况，而时序差分方法借助试验才能知道。

蒙特卡洛方法和时序差分方法的不同在于，蒙特卡洛方法进行了完整的采样来获取长期的回报值，因而在价值估计上会有更小的偏差，但是也正因为收集了完整的信息，所以价值的方差会更大，原因在于其基于试验的采样得到，和真实的分布有差距，不充足的交互导致较大方差。而时序差分方法则相反，因为它只考虑了前一步的回报值，其他都是基于之前的估计值，因而其价值估计相对来说具有偏差大方差小的特点。

三者的联系：对于  $TD(\lambda)$  方法，如果  $\lambda = 0$ ，那么此时等价于时序差分方法，即只考虑下一个状态；如果  $\lambda = 1$ ，等价于蒙特卡洛方法，即考虑  $T - 1$  个后续状态直到整个试验结束。

## 第 4 章面试题解答

**4-1**

首先我们的目的是最大化奖励函数，即调整  $\theta$ ，使得期望回报最大，可以用公式表示如下：

$$J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_t r(s_t, a_t) \right]$$

其中  $\tau$  表示从开始到结束的一条完整轨迹。通常对于最大化问题，我们可以使用梯度上升算法找到最大值，即

$$\theta^* = \theta + \alpha \nabla J(\theta)$$

所以我们仅仅需要计算并更新  $\nabla J(\theta)$ ，也就是计算奖励函数  $J(\theta)$  关于  $\theta$  的梯度，也就是策略梯度，计算方法如下：

$$\nabla_{\theta} J(\theta) = \int \nabla_{\theta} p_{\theta}(\tau) r(\tau) d\tau = \int p_{\theta} \nabla_{\theta} \log p_{\theta}(\tau) r(\tau) d\tau = E_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$$

接着我们继续展开，对于  $p_{\theta}(\tau)$ ，即  $p_{\theta}(\tau|\theta)$ ：

$$p_{\theta}(\tau|\theta) = p(s_1) \prod_{t=1}^T \pi_{\theta}(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

取对数后为：

$$\log p_{\theta}(\tau|\theta) = \log p(s_1) + \sum_{t=1}^T \log \pi_{\theta}(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

继续求导：

$$\nabla \log p_{\theta}(\tau|\theta) = \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t|s_t)$$

代入第 3 个式子，可以将其化简为：

$$\begin{aligned} \nabla_{\theta} J(\theta) &= E_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)] \\ &= E_{\tau \sim p_{\theta}} [(\nabla_{\theta} \log \pi_{\theta}(a_t|s_t)) (\sum_{t=1}^T r(s_t, a_t))] \\ &= \frac{1}{N} \sum_{i=1}^N [(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t}|s_{i,t})) (\sum_{t=1}^N r(s_{i,t}, a_{i,t}))] \end{aligned}$$

## 4-2

(1) 增加基线：为了防止所有奖励都为正，从而导致每一个状态和动作的变换，都会使得每一个变换的概率上升，我们把奖励减去一项  $b$ ，称  $b$  为基线。当减去  $b$  以后，就可以让奖励  $R(\tau^n) - b$  有正有负。如果得到的总奖励  $R(\tau^n)$  大于  $b$ ，就让它的概率上升。如果总奖励小于  $b$ ，就算它是正的，值很小也是不好的，就需要让它的概率下降。如果总奖励小于  $b$ ，就要让采取这个动作的奖励下降，这样也符合常理。但是使用基线会让本来奖励很大的“动作”的奖励变小，降低更新速率。

(2) 指派合适的分数：首先，原始权重是整个回合的总奖励。现在改成从某个时间点  $t$  开始，假设这个动作是在时间点  $t$  被执行的，那么从时间点  $t$ ，一直到游戏结束所有奖励的总和，才真的代表这个动作是好的还是不好的；接下来我们再进一步，把未来的奖励打一个折扣，这里我们称由此得到的奖励的和为折扣回报。

(3) 综合以上两种技巧，我们将其统称为优势函数，用  $A$  来代表优势函数。优势函数取决于状态和动作，即我们需计算的是在某一个状态  $s$  采取某一个动作  $a$  的时候，优势函数有多大。

## 第 5 章面试题解答

### 5-1

使用另外一种分布，来逼近所求分布的一种方法，算是一种期望修正的方法，公式如下：

$$\int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x \sim q}[f(x)\frac{p(x)}{q(x)}] = E_{x \sim p}[f(x)]$$

我们在已知  $q$  的分布后，可以使用上式计算出从  $p$  分布的期望值。也就可以使用  $q$  来对  $p$  进行采样了，即重要性采样。

### 5-2

我可以一句话概括两者的区别，即生成样本的策略（价值函数）和网络参数更新时的策略（价值函数）是否相同。具体来说，同策略，生成样本的策略（价值函数）与网络更新参数时使用的策略（价值函数）相同。Sarsa 算法就是同策略的，其基于当前的策略直接执行一次动作，然后用价值函数的值更新当前的策略，因此生成样本的策略和学习时的策略相同，算法为同策略算法。该算法会遭遇探索-利用窘境，仅利用目前已知的最优选择，可能学不到最优解，不能收敛到局部最优，而加入探索又降低了学习效率。 $\varepsilon$ -贪

心算法是这种矛盾下的折中，其优点是直接了当、速度快，缺点是不一定能够找到最优策略。异策略，生成样本的策略（价值函数）与网络更新参数时使用的策略（价值函数）不同。例如，Q 学习算法在计算下一状态的预期奖励时使用了最大化操作，直接选择最优动作，而当前策略并不一定能选择到最优动作，因此这里生成样本的策略和学习时的策略不同，即异策略算法。

### 5-3

近端策略优化算法借鉴了信任区域策略优化算法，通过采用一阶优化，在采样效率、算法表现以及实现和调试的复杂度之间取得了新的平衡。这是因为近端策略优化算法会在每一次迭代中尝试计算新的策略，让损失函数最小化，并且保证每一次新计算出的策略能够和原策略相差不大。换句话说，其为在避免使用重要性采样时由于在  $\theta$  下的  $p_\theta(a_t|s_t)$  与在  $\theta'$  下的  $p_{\theta'}(a_t|s_t)$  差太多，导致重要性采样结果偏差较大而采取的算法。

## 第 6 章面试题解答

### 6-1

深度 Q 网络是基于深度学习的 Q 学习算法，其结合了价值函数近似与神经网络技术，并采用了目标网络和经验回放技巧进行网络的训练。

### 6-2

在深度 Q 网络中某个动作价值函数的更新依赖于其他动作价值函数。如果我们一直更新价值网络的参数，会导致更新目标不断变化，也就是我们在追逐一个不断变化的目标，这样势必会不太稳定。为了解决基于时序差分的网络中，优化目标  $Q_\pi(s_t, a_t) = r_t + Q_\pi(s_{t+1}, \pi(s_{t+1}))$  左右两侧会同时变化使得训练过程不稳定，从而增大回归难度的问题，目标网络选择将优化目标的右边即  $r_t + Q_\pi(s_{t+1}, \pi(s_{t+1}))$  固定，通过改变优化目标左边的网络参数进行回归。对于经验回放，其会构建一个回放缓冲区，用来保存许多数据，每一个数据的内容包括：状态  $s_t$ 、采取的动作  $a_t$ 、得到的奖励  $r_t$ 、下一个状态  $s_{t+1}$ 。我们使用  $\pi$  与环境交互多次，把收集到的数据都放到回放缓冲区中。当回放缓冲区“装满”后，就会自动删去最早进入缓冲区的数据。在训练时，对于每一轮迭代都有相对应的批量（与我们训练普通网络一样，通过采样得到），然后用这个批量中的数据去更新 Q 函数。即 Q 函数在采样和训练的时候会用到过去的经验数据，也可以消除样本之间的相关性。

### 6-3

整体来说，从名称就可以看出，两者的目标价值以及价值的更新方式基本相同。但有如下不同点：

(1) 首先，深度 Q 网络将 Q 学习与深度学习结合，用深度网络来近似动作价值函数，而 Q 学习则是采用表格进行存储。

(2) 深度 Q 网络采用了经验回放的技巧，从历史数据中随机采样，而 Q 学习直接采用下一个状态的数据进行学习。

### 6-4

随机性策略表示为某个状态下动作取值的分布，确定性策略在每个状态只有一个确定的动作可以选。从熵的角度来说，确定性策略的熵为 0，没有任何随机性。随机性策略有利于我们进行适度的探索，确定性策略不利于进行探索。

### 6-5

在神经网络中通常使用随机梯度下降法。随机的意思是我们随机选择一些样本来增量式地估计梯度，比如常用的批量训练方法。如果样本是相关的，就意味着前后两个批量很可能也是相关的，那么估计的梯度也会呈现出某种相关性。但是在极端条件下，后面的梯度估计可能会抵消掉前面的梯度估计量，从而使训练难以收敛。

## 第 7 章面试题解答

### 7-1

深度 Q 网络有 3 个经典的变种：双深度 Q 网络、竞争深度 Q 网络、优先级双深度 Q 网络。

- (1) 双深度 Q 网络：将动作选择和价值估计分开，避免 Q 值被过高估计。
- (2) 竞争深度 Q 网络：将 Q 值分解为状态价值和优势函数，得到更多有用信息。
- (3) 优先级双深度 Q 网络：将经验池中的经验按照优先级进行采样。

### 7-2

深度 Q 网络由于总是选择当前最优的动作价值函数来更新当前的动作价值函数，因此存在过估计问题（估计的价值函数值大于真实的价值函数值）。为了解耦这两个过程，双深度 Q 网络使用两个价值网络，一个网络用来执行动作选择，然后用另一个网络的价值函数对应的动作值更新当前网络。

### 7-3

对于  $\mathbf{Q}(s, a)$ ，其对应的状态由于为表格的形式，因此是离散的，而实际的状态大多不是离散的。对于 Q 值  $\mathbf{Q}(s, a) = V(s) + \mathbf{A}(s, a)$ 。其中的  $V(s)$  是对于不同的状态都有值， $\mathbf{A}(s, a)$  对于不同的状态都有不同的动作对应的值。所以本质上，我们最终的矩阵  $\mathbf{Q}(s, a)$  是将每一个  $V(s)$  加到矩阵  $\mathbf{A}(s, a)$  中得到的。但是有时我们更新时不一定会将  $V(s)$  和  $\mathbf{Q}(s, a)$  都更新。我们将其分成两个部分后，就不需要将所有的状态-动作对都采样一遍，我们可以使用更高效的估计 Q 值的方法将最终的  $\mathbf{Q}(s, a)$  计算出来。

## 第 9 章面试题解答

### 9-1

A3C 是异步优势演员-评论员算法，其中，评论员学习价值函数，同时有多个演员并行训练并且不时与全局参数同步。A3C 旨在并行训练，是同策略算法。

### 9-2

- (1) 相比以价值函数为中心的算法，演员-评论员算法应用了策略梯度的技巧，这能让它在连续动作或者高维动作空间中选取合适的动作，而 Q 学习做这件事会很困难。
- (2) 相比单纯策略梯度，演员-评论员算法应用了 Q 学习或其他策略评估的做法，使得演员-评论员算法能进行单步更新而不是回合更新，比单纯的策略梯度的效率要高。

### 9-3

下面是异步优势演员-评论员算法的大纲，由于其为异步多线程算法，我们只对其中某一单线程进行分析。

- (1) 定义全局参数  $\theta$  和  $w$  以及特定线程参数  $\theta'$  和  $w'$ 。(2) 初始化时间步  $t = 1$ 。(3) 当  $T \leq T_{\max}$ 。
  - 重置梯度： $d\theta = 0$  并且  $dw = 0$ 。
  - 将特定于线程的参数与全局参数同步： $\theta' = \theta$  以及  $w' = w$ 。
  - 令  $t_{\text{start}} = t$  并且随机采样一个初始状态  $s_t$ 。
  - 当 ( $s_t \neq$  终止状态) 并且  $t - t_{\text{start}} \leq t_{\max}$ 。
    - 根据当前线程的策略选择当前执行的动作  $a_t \sim \pi_{\theta'}(a_t | s_t)$ ，执行动作后接收奖励  $r_t$  然后转移到下一个状态  $s_{t+1}$ 。
    - 更新  $t$  以及  $T$ ： $t = t + 1$  并且  $T = T + 1$ 。
  - 初始化保存累积奖励估计值的变量。
  - 对于  $i = t_1, \dots, t_{\text{start}}$ 。
    - $r \leftarrow \gamma r + r_i$ ；这里的  $r$  是  $G_i$  的蒙特卡洛估计。
    - 累积关于参数  $\theta'$  的梯度： $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi_{\theta'}(a_i | s_i) (r - V_{w'}(s_i))$ 。
    - 累积关于参数  $w'$  的梯度： $dw \leftarrow dw + \partial(r - V_{w'}(s_i))^2 / \partial w'$ 。
  - 分别使用  $d\theta$  以及  $dw$  异步更新  $\theta$  以及  $w$ 。

### 9-4

演员是策略模块，输出动作；评论员是判别器，用来计算价值函数。

### 9-5

评论员衡量当前决策的好坏。结合策略模块，当评论员判别某个动作的选择是有益的时候，策略就更新参数以增大该动作出现的概率，反之减小该动作出现的概率。

### 9-6

优势函数的计算公式为  $A(s, a) = Q(s, a) - V(s) = r + \gamma V(s') - V(s)$ ，其可以定量地表示选择动作  $a$  的优势。即当动作  $a$  低于价值函数的平均值的时候，优势函数为负值；反之为正值。其是一个标量，具体来说：

- (1) 如果  $A(s, a) > 0$ ，梯度被推向正方向；
- (2) 如果  $A(s, a) < 0$ ，即我们的动作比该状态下的平均值还差，则梯度被推向反方向。

这样就需要两个价值函数，所以可以使用时序差分方法做误差估计： $A(s, a) = r + \gamma V(s') - V(s)$ 。

## 第 12 章面试题解答

### 12-1

深度确定性策略梯度算法使用演员-评论员结构，但是输出的不是动作的概率，而是具体动作，其可以用于连续动作的预测。优化的目的是将深度 Q 网络扩展到连续的动作空间。另外，其含义如其名：

- (1) 深度是因为用了深度神经网络；
- (2) 确定性表示其输出的是一个确定的动作，可以用于连续动作的环境；
- (3) 策略梯度代表的是它用到的是策略网络。强化算法每个回合就会更新一次网络，但是深度确定性策略梯度算法每个步骤都会更新一次策略网络，它是一个单步更新的策略网络。

### 12-2

异策略算法。(1) 深度确定性策略梯度算法是优化的深度 Q 网络，其使用了经验回放，所以为异策略算法。(2) 因为深度确定性策略梯度算法为了保证一定的探索，对输出动作加了一定的噪声，行为策略不再是优化的策略。

### 12-3

分布的分布式深度确定性策略梯度算法 (distributed distributional deep deterministic policy gradient, D4PG)，相对于深度确定性策略梯度算法，其优化部分如下。

- (1) 分布式评论员：不再只估计 Q 值的期望值，而是估计期望 Q 值的分布，即将期望 Q 值作为一个随机变量来估计。
- (2)  $N$  步累计回报：计算时序差分误差时，D4PG 计算的是  $N$  步的时序差分目标值而不仅仅只有一步，这样就可以考虑未来更多步骤的回报。
- (3) 多个分布式并行演员：D4PG 使用  $K$  个独立的演员并行收集训练数据并存储到同一个回放缓冲区中。
- (4) 优先经验回放 (prioritized experience replay, PER)：使用一个非均匀概率从回放缓冲区中进行数据采样。