18 深度强化学习应用实践技巧

之前的章节向读者展现了深度强化学习的主要知识点、强化学习算法的主要类别和算法实现，以及为了便于理解深度强化学习应用而讲解的几个实践项目。然而，由于如之前强化学习的挑战一章中提到的低样本效率、不稳定性等问题，初学者想要较好地部署这些算法到自己的应用中还是有一定困难的。因此，在这一章，从数学分析和实践经验的角度，我们细致地总结了一些在深度强化学习应用实践中常用的技巧和方法。这些方法或小窍门涉及了算法实现阶段和训练调试阶段，用来帮助读者避免陷入一些实践上的困境。这些经验上的技巧有时可以产生显著效果，但不总是这样。这是由于深度强化学习模型的复杂性和敏感性造成的，而有时需要同时使用多个技巧。如果在某一个项目上卡住时，大家也可以从这一章中得到一些解决方案上的启发。

18.1 概览：如何应用深度强化学习

深度学习通常被认为是“黑盒”方法。尽管它实际上并不是“黑盒”，但是它有时会表现得不稳定且会产生不可预测的结果。在深度强化学习中，由于强化学习的基本过程需要智能体从与环境交互的动态过程中的奖励信号而不是标签中学习，这个问题变得更加严重。这是与有监督学习的情况不同的。强化学习中的奖励函数可能只包含不完整或者局部的信息，而智能体使用自举（Bootstrapping）学习方法时往往在追逐一个变化的目标。此外，深度强化学习中经常用到不止一个深度神经网络，尤其是在那些较为高等或者最近提出的方法中。这都使得深度强化学习算法可能表现得不稳定且对超参数敏感。以上问题使得深度强化学习的研究和应用困难重重。由于这个原因，我们在这里介绍一些实现深度强化学习中常用的技巧和建议。

首先，你需要知道一个强化学习算法是否可以用于解决某一个特定的问题，而且显然不是每个算法都对所有任务适用。我们经常需要仔细考虑强化学习本身是否可以用于解决某个任务。总

体来说，强化学习可以用于连续决策制定问题，而这类问题通常可以用马尔可夫（Markov）过程来描述或近似。一个有标签数据的预测任务通常不需要强化学习算法，而监督学习方法可能更直接和有效。强化学习任务通常包括至少两个关键要素：（1）环境，用来提供动态过程和奖励信号；（2）智能体，由一个策略控制，而这个策略是通过强化学习训练得到的。在之前的几个章节中强化学习算法被用来解决像 OpenAI Gym 这类环境中的任务。在这些实验中，你不需要过多关心环境，因为它们已经被设计好且经过标准化和正则化。然而，在应用章节中介绍的几个项目则需要人为定义环境，并运用强化学习算法去使智能体正常工作。

总的来说，应用深度强化学习算法有以下几个阶段。

1. **简单测试阶段**：你需要使用对其正确性和准确性有高置信度的模型，包括强化学习算法，如果是一个新的任务，用它来探索环境（甚至使用一个随机策略）或者逐步验证你将在最终模型上做的延伸，而不是直接使用一个复杂的模型。你需要快速进行实验来检测环境和模型基本设置中可能的问题，或者至少让你自己熟悉这个要解决的任务，这会给你在之后的过程中提供一些启发，有时也会暴露出一些需要考虑的极端情况。

2. **快速配置阶段**：你应该对模型设置做快速测试，来评估其成功的可能性。如果有错误，尽可能多地可视化学习过程，并在你无法直接从数字上得到潜在关系的时候使用一些统计变量（方差、均值、平均差值、极大极小值等）。这一步应当在简单测试阶段后开始，然后逐步增加新模型的复杂度。如果你无法百分之百确定更改的有效性，你应当每一次都进行测试。

3. **部署训练阶段**：在你仔细确认过模型的正确性后，你可以开始大规模部署训练了。由于深度强化学习往往需要较大量的样本去训练较长时间，我们鼓励你使用并行训练方式、使用云服务器（如果你自己没有服务器的话）等，来加速对最终模型的大规模训练。有时这一阶段是和第二阶段交替进行的，因而这一步在实践中可能会花费较长时间。

在下面几个小节中，我们将分几部分介绍应用深度强化学习的技巧。

18.2　实现阶段

- **从头实现一些基本的强化学习算法**。对于深度强化学习领域的初学者而言，从头实现一些基本的强化学习算法并调试这些算法直到它们最终正确运行，是很好的练习。Deep Q-Networks 作为一种基于价值函数的算法，是值得去自己实现的。连续动作空间、策略梯度和 Actor-Critic 算法也是刚开始学习强化学习算法实现时很好的选择。这个过程会需要你理解强化学习算法实现中的每一行代码，给你一个强化学习过程的整体感觉。刚开始，你不需要一个复杂的大规模任务，而是一个相对简单的可以快速验证的任务，比如那些 OpenAI Gym 环境。在实现这些基本算法的时候，你应当基于一种公用的结构并且使用一种深度学习框架（比如 TensorFlow、PyTorch 等），并逐步扩展到更加复杂的任务上，同时使用更加高级的技术（比如优先经验回放等）。这会显著地加速你随后将不同的深度强化学习算法应用于其他项目的进程。如果你在实现过程中遇到一些问题，你可以参考其他人的实现方法（比如本

书提供的强化学习算法实现指南）或者通过网络查找你遇到的问题。绝大多数问题都已经被他人所解决。

- **适当地实现论文细节。**在你熟悉了这些基本的强化学习算法后，就可以开始实现和测试一些在文献中的方法。通常强化学习算法的研究论文中包含很多实现细节，而有时这些细节在不同论文中不是一致的。所以，当你实现这些方法的时候，不要过拟合到论文细节上，而是去理解论文作者为何在这些特定情形下选择使用这些技巧。举一个典型的例子，在多数文章中，实验部分中神经网络的结构细节包括隐藏层的维数和层数、各个超参数的数值等。这些都或在论文主体、或在补充材料中提到。你不需要在自己的实现版本中严格遵循这些实现细节，而且你很可能甚至跟原文用不一样的环境来对方法进行测试。比如，在深度决定性策略梯度（Deep Deterministic Policy Gradient）算法的论文中，作者建议使用 Ornstein-Uhlenbeck（OU）噪声来进行探索。然而，实践中，有时很难说 OU 噪声是否比高斯噪声更好，而这往往在很大程度上依赖于具体任务。另一个例子是，在 Vinyals et al. (2019) 关于 AlphaStar 的工作中，Vanilla TD(λ) 方法被证实比其他更高级的离线策略（Off-Policy）修正方法 V-Trace (Espeholt et al., 2018) 更有效。因此，如果这些技巧不足够通用，那么它们可能不值得你花精力去实现。相比而言，一些微调方法可能对具体任务有更好的效果。然而，如上所述，理解作者在这些情况下为何使用这些技巧则是更关键和有意义的。当你采取论文中的某些想法并将其应用到你自己的方法中时，这些建议可能更有意义，因为有时对于你自己的具体情况，可能不是论文中的主要想法，而是某些具体技巧或操作对你帮助最大。
- **如果你在解决一个具体任务，先探索一下环境。**你应当检查一下环境的细节，包括观察量和动作的性质，如维度、值域、连续或离散值类型等。如果环境观察量的值在一个很大的有效范围内或者是未知范围的，你就应该把它的值归一化。比如，如果你使用 Tanh 或者 Sigmoid 作为激活函数，较大的输入值将可能使第一个隐藏层的节点饱和，而这训练开始时将导致较小的梯度值和较慢的学习速度。此外，你应当为强化学习选择好的输入特征，这些特征应当包含环境的有用信息。你也可以用能进行随机动作选取的智能体来探索环境并可视化这个过程，以找到一些极端情况。如果环境是你自己搭建的，这一步可能很重要。
- **给每一个网络选取一个合适的输出激活函数。**你应当根据环境来对动作网络选择一个合适的输出激活函数。比如，常用的像 ReLU 可能从计算时间和收敛表现上都对隐藏层来说可以很好地工作，但是它对有负值的动作输出范围来说可能是不合适的。最好将策略输出值的范围跟环境的动作值域匹配起来，比如对于动作值域 $(-1, 1)$ 在输出层使用 Tanh 激活函数。
- **从简单例子开始逐渐增加复杂度。**你应当从比较清晰的模型或环境开始测试，然后逐步增加新的部分，而不是一次将所有的模块组合起来测试和调试。在实现过程中不断进行测试，除非你是这个领域的专家并且很幸运，否则你不应当期望一个复杂的模型可以一次实现成功并得到很好的结果。
- **从密集奖励函数开始。**奖励函数的设计可以影响学习过程中优化问题的凸性，因此你应当从一个平滑的密集奖励函数开始尝试。比如，在第 16 章中定义的机器人抓取任务中，我们

用一个密集奖励函数来开始机器人学习，这个函数是从机器人夹具到目标物体之间距离的负数。这可以保证值函数网络和策略网络能够在一个较为光滑的超平面上优化，从而显著地加速学习过程。一个稀疏奖励可以被定义为一个简单的二值变量，用来表示机器人是否抓取了目标物体，而在没有额外信息的情况下这对机器人来说可能很难进行探索和学习。

- **选择合适的网络结构**。尽管在深度学习中经常见到一个有几十层网络和数十亿参数的网络，尤其在像计算机视觉 (He et al., 2016) 和自然语言处理 (Jaderberg et al., 2015) 领域。对于深度强化学习而言，神经网络深度通常不会太深，超过 5 层的神经网络在强化学习应用中不是特别常见。这是由于强化学习算法本身的计算复杂度造成的。因此，除非环境有很大的规模而且你有几十上百个 GPU 或者 TPU 可以使用，否则你一般不会在深度强化学习中用一个 10 层及以上的网络，它的训练将会非常困难。这不仅是计算资源上的限制，而且这也与深度强化学习由于缺失监督信号而导致的不稳定性和非单调表现增长有关。在监督学习中，如果网络相比于数据而言足够大，它可以过拟合到数据集上，而在深度强化学习中，它可能只是缓慢地收敛甚至是发散，这是因为探索和利用之间的强关联作用。网络大小的选择经常是依据环境状态空间和动作空间而定的。一个有几十个状态动作组合的离散环境可能可以用一个表格方法，或者一个单层或两层的神经网络解决。更复杂的例子如第 13 章和第 16 章中介绍的应用，通常有几十维的连续状态和动作空间，这就需要可能大于 3 层的网络，但是相比于其他深度学习领域中的巨型网络而言，这仍旧是很小的规模。

  对于网络的结构而言，文献中很常见的有多层感知机（Multi-Layer Perceptrons，MLPs）、卷积神经网络（CNNs）和循环神经网络（RNNs）。更为高级和复杂的网络结构很少用到，除非对模型微调方面有具体要求或者一些其他特殊情况。一个低维的矢量输入可以用一个多层感知机处理，而基于视觉的策略经常需要一个卷积神经网络主干来提前提取信息，要么与强化学习算法一起训练，要么用其他计算机视觉的方法进行预训练。也有其他情况，比如将低维的矢量输入和高维的图像输入一起使用，实践中通常先采用从高维输入中提取特征的主干再与其余低维输入并联的方法。循环神经网络可以用于不是完全可观测的环境或者非马尔可夫过程，最优的动作选择不仅依赖当前状态，而且依赖之前状态。以上是实践中对策略和价值网络都有效的经验指导。有时策略和价值网络可能构成一种非对称的 Actor-Critic 结构，因而它们的状态输入是不同的，这可以用于价值网络只用作训练中策略网络的指导，而在动作预测时不再可以使用价值网络的情况。

- **熟悉你所用的强化学习算法的性质**。举例来说，像 PPO 或 TRPO 类的基于信赖域的方法可能需要较大的批尺寸来保证安全的策略进步。对于这些信赖域方法，我们通常期待策略表现稳定的进步，而非在学习曲线上某些位置突然有较大下降。TRPO 等信赖域方法需要用一个较大批尺寸的原因是，它需要用共轭梯度来近似 Fisher 信息矩阵，这是基于当前采样到的批量样本计算的。如果批尺寸太小或者是有偏差的，可能对这个近似造成问题，并且导致对 Fisher 信息矩阵（或逆 Hessian 乘积）的近似不准确而使学习表现下降。因此，实践中，算法 TRPO 和 PPO 中的批尺寸需要被增大，直到智能体有稳定进步的学习表现为止。

所以，TRPO 有时也无法较好地扩展到大规模的网络或较深的卷积神经网络和循环神经网络上。DDPG 算法则通常被认为对超参数敏感，尽管它被证明对许多连续动作空间的任务很有效。当把它应用到大规模或现实任务 (Mahmood et al., 2018) 上时，这个敏感性会更加显著。比如，尽管在一个简单的模拟测试环境中通过彻底的超参数搜索可以最终找到一个最优的表现效果，但是在现实世界中的学习过程由于时间和资源上的限制可能不允许这种超参数搜索，因此 DDPG 相比与其他 TRPO 或 SAC 算法可能不会有很好的效果。另一方面，尽管 DDPG 算法起初是设计用来解决有连续值动作的任务，这并不意味着它不能在离散值动作的情况下工作。如果你尝试将它应用到有离散值动作的任务上，那么需要使用一些额外的技巧，比如用一个有较大 t 值的 Sigmoid(tx) 输出激活函数并且将其修剪成二值化的输出，还得保证这个截断误差比较小，或者你可以直接使用 Gumbel-Softmax 技巧来更改确定性输出为一个类别的输出分布。其他算法也可以有相似处理。

- **归一化值处理**。总体来说，你需要通过缩放而不是改变均值来归一化奖励函数值，并且用同样的方式标准化值函数的预测目标值。奖励函数的缩放基于训练中采样的批样本。只做值缩放（即除以标准差）而不做均值平移（为得到零均值而减去统计均值）的原因是，均值平移可能会影响到智能体的存活意愿。这实际上与整个奖励函数的正负号有关，而且这个结论只适用于你使用“Done”信号的情况。其实，如果你事先没有用“Done”信号来终止片段，那么，你可以使用均值平移。考虑以下一种情况，如果智能体经历了一个片段，而“Done=True”信号在最大片段长度以内发生，那么假如我们认为智能体仍旧存活，则这个“Done”信号之后的奖励值实际为 0。如果这些为 0 的奖励值总体上比之前的奖励值高（即之前的奖励值基本是负数），那么智能体会倾向于尽可能早地结束片段，以最大化整个片段内的奖励。相反，如果之前的奖励函数基本是正值，智能体会选择“活”得更久一些。如果我们对奖励值采取均值平移方式，它会打破以上情形中智能体的存活意愿，从而使得智能体即使在奖励值基本为正时不会选择存活得更久，而这会影响训练中的表现。归一化值函数的目标也是相似的情况。举例来说，一些基于 DQN 的算法的平均 Q 值会在学习过程中意外地不断增大，而这是由最大化优化公式中对 Q 值的过估计造成的。归一化目标 Q 值可以缓解这个问题，或者使用其他的技巧如 Double Q-Learning。
- **一个关于折扣因子的小提示**。你可以根据折扣因子 γ 对单步动作选择的有效时间范围有一个大致感觉：$1+\gamma+\gamma^2+\cdots=1/(1-\gamma)$。根据该式，对于 $\gamma=0.99$，我们经常可以忽略 100 个时间步后的奖励。用这个小技巧可以加速你设置参数时的过程。
- **Done 信号只在终止状态时为真**。对于初学者来说，深度强化学习中有一些很容易忽略的细微差别，而片段式强化学习中的“Done”信号就是其中一个。这些细微的差异可能使得实践中即使是相同算法的不同实现也会有截然不同的表现。在片段式强化学习中，“Done”信号被广泛用于结束一个片段，而它是环境状态的一个函数，只要智能体到达终止状态，它就被设置为真。注意，这里终止状态被定义为指示智能体已经完成片段的情况，要么成功要么失败，而不是任意一个到达时间限度或最大片段长度的状态。将“Done”信号的值只

在状态为终止状态时设为真不是一个平庸的问题。举例来说，如果一个任务是操控机械臂到达空间中某个具体的位置，这个“Done”信号只应当在机械臂确实到达这个位置时为真，而不是到达默认片段最大长度等情况下。为了理解这个差异，我们需要知道在强化学习中有些环境，时间长度是无穷的，有些是有限的，而在采样过程中，算法经常是对有限长度的轨迹做处理的。有两种常用的实现方式，一是设置最大片段长度，二是使用“Done”信号作为环境的反馈来通过跳开循环以终止片段。当使用“Done”信号作为采样过程中的中断点时，它不应当在片段由于到达最大长度的时候设为真，而只应在终止状态到达时为真。还是前面的例子，若一个机械臂在非目标点的任意其他点由于到达了片段最大长度而结束了这个轨迹，同时设置了“Done”信号为真，则会对学习过程产生消极影响。具体来说，以 PPO 算法为例，从状态 S_t 累计的奖励值被用来估计该状态的价值 $V(S_t)$，而一个终止状态的价值为 0。如果在非终止状态时“Done”信号的值为真，那么该状态的值被强制设为 0 了，而实际上它可能不应该为 0。这会在价值网络估计之前状态值的时候让其产生混淆，从而阻碍学习过程。

- **避免数值问题。**对于编程实践中的除法，如果使用不当可能会产生无穷大的数值。两个技巧可以解决这类问题：一个是对正数值的情况使用指数缩放 $a/b = \exp(\log(a) - \log(b))$；另一个方法是对于非负分母加上一个小量，如 $a/b \approx a/(b + 10^{-6})$。
- **注意奖励函数和最终目标之间的分歧。**强化学习经常被用于一个有最终目标的具体任务，而通常需要人为设计一个与最终目标一致的奖励函数来便于智能体学习。在这个意义上说，奖励函数是目标的一种量化形式，这也意味着它们可能是两个不同的东西。在某些情况下它们之间会有分歧。因为一个强化学习智能体能够过拟合到你为任务所设置的奖励函数上，而你可能发现训练最终策略在达成最终目标上与你所期望的不同。这其中一个最可能的原因是奖励函数和最终目标之间的分歧。在多数情况下，奖励函数倾向于最终的任务目标是容易的，但是设计一个奖励函数与最终目标在所有极端情况下都始终一致，是不平庸的。你应该做的是尽可能减少这种分歧，来保证你设计的奖励函数能够平滑地帮助智能体达到最终真实目标。
- **奖励函数可能不总是对学习表现的最好展示。**人们通常在学习过程中展示奖励函数值（有时用移动平均，有时不用）来表示一个算法的能力。然而，如同上面所说，最终目标和你所定义的奖励函数之间可能有分歧，这使得一个较高奖励的状态可能对应一个在达成最终目标方面较差的情况，或者至少没有显式地表现出该状态与最优状态之间的关系。由于这个原因，我们总需要在使用强化学习和展示结果时考虑这种分歧的可能性。所以，在文献 (Fu et al., 2018) 中很常见到，有的学习表现不是用平滑后的片段内奖励（这也依赖奖励函数的设计）来评估和展示的，而是用一个对这个任务更具体的度量方式，比如图 18.1 所示的机器人学习任务中，用机器夹具跟目标点的距离来实现位置到达或用物块跟目标的距离来实现物块推动。夹具与物体间的距离，或者是否物块被抓取，这些都是对任务目标的真实度量方式。所以，这些度量可以用来展示任务学习效果，从而更好地体现任务最终目标是否

到达。这对于最终目标跟人为设计的奖励函数有偏差的情况很有用，如果你想比较多个不同个奖励函数，那么这些额外的度量也很关键。

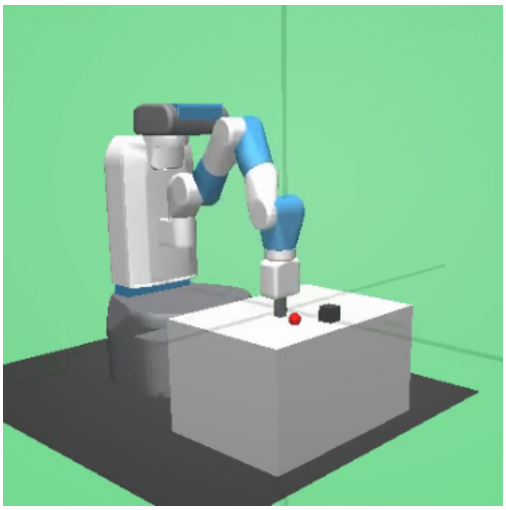

图 18.1 OpenAI Gym 中的 FetchPush 环境。对这个环境而言，使用物体到目标位置的最终距离比奖励函数值能更好地衡量对所学策略的表现，因为它是对任务整体目标的最直接表示。然而奖励函数可能被设置为包含一些其他因素，如夹具到物体的距离等（见彩插）

- **非马尔可夫情况。**如之前章节所述，这本书所介绍的绝大多数理论结果都基于马尔可夫过程的假设或者状态的马尔可夫性质。马尔可夫性质不仅简化了问题和推导，更重要的是它使得连续决策问题可以描述，而且可以用迭代的方式解决它，还能得到简洁的解决方法。然而，实践中，马尔可夫过程的假设不总是成立。举例来说，如图 18.2 所示，Gym 环境中的 Pong 游戏就不满足马尔可夫过程在智能体选取最优动作时对状态所做的假设。我们需要记住马尔可夫性质是状态或环境的性质，因此它是由状态的定义决定的。非马尔可夫决策过程和部分可观测马尔可夫过程（POMDP）的差异有时是细微的。比如，如果一个在上述游戏中状态被定义为同时包含小球的位置和速度信息（假设小球运动没有加速度），而观察量只有位置，那么这个环境是 POMDP 而不是非马尔可夫过程。然而，Pong 游戏的状态通常被认为是每一个时间步静态帧，那么当前状态只包含小球的位置而没有智能体能够做出最优动作选择的所有信息，比如，小球速度和小球运动方向也会影响最优动作。所以这种情况下它是一个非马尔可夫环境。一种提供速度和运动方向信息的方法是使用历史状态，而这违背了马尔可夫过程下的处理方法。所以，如 DQN (Mnih et al., 2015) 原文，堆叠帧可以以一种近似的 MDP 来解决 Pong 任务。如果我们把所有的堆叠帧看作一个单一状态，并且假设堆叠帧可以包含做出最优动作选择的所有信息，那么这个任务实际上仍旧遵从马尔可夫过程假设。毕竟在所有的模拟环境中，过程都是离散的，而不像现实世界中，有时间尺度上的连续性，我们经常可以用这种转化方式来把一个非马尔可夫过程看作一个马尔可夫

过程。除了像 DQN 原文中使用堆叠帧，循环神经网络（RNN）(Heess et al., 2015) 或更高级的长短期记忆（LSTM）方法也可以用于以历史记忆进行决策的情况，来解决非马尔可夫过程的问题。

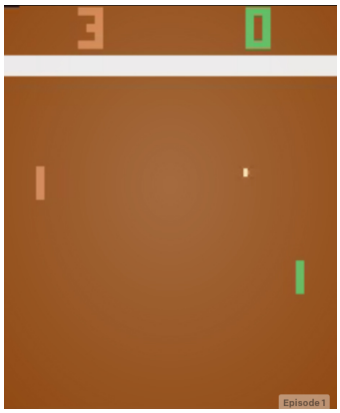

图 18.2　Gym 中的 Pong-v0 游戏：由于小球的速度无法在单一帧中捕捉，这个任务是非马尔可夫的，用堆叠帧作为观察量可以解决它

18.3　训练和调试阶段

- **初始化很重要。**深度强化学习方法通常要么以在线策略（On-Policy）方式用每个片段内的样本更新策略，要么使用离线策略（Off-Policy）中动态的回放缓存（Replay Buffer），这个缓存包含随时间变化的多样性样本。这使得深度强化学习不同于监督学习，监督学习是从一个固定的数据集中学习，因而学习样本的顺序不是特别重要。然而，在深度强化学习中，策略的初始化可以影响随后可能的探索范围，并决定存入缓存的后续样本或直接用于更新的样本，因此它会影响整个学习表现。从一个随机策略开始会导致较大的概率有更多样的样本，这对于训练开始阶段是很好的。但随着策略的收敛和进步，探索的范围逐渐收窄，而近趋于当前策略所生成的轨迹。对于权重参数的初始化而言，总体上来说使用较高级的方法如 Xavier 初始化 (Glorot et al., 2010) 或正交初始化 (Saxe et al., 2013) 会较好，这样可以避免梯度消失或梯度爆炸，并且对多数深度学习情况都有较稳定的学习表现。
- **向程序中添加有用的探针。**深度学习往往要处理大量的数据，而这其中有一些隐藏的操作是我们可能不总清楚，尤其是当我们对模型不熟悉的时候。通常的报错系统可能不是针对其中一些错误的，尤其是逻辑错误。模型中类似的潜在问题将会是很危险和难以察觉的。比如，有时在深度强化学习中，你只关注奖励函数，但是也应当可视化损失函数值的变化来了解其他函数的拟合情况，比如价值函数，或者随机分布策略的熵来了解当前的探索状态。如果策略输出分布熵过早下降，那么基本上表明智能体不能通过当前策略探索到更有用的样

本。这可以通过使用熵奖励或 KL 散度惩罚项来缓解，如柔性 Actor-Critic（Soft Actor-Critic，SAC）等算法使用了适应性熵类自动解决这类问题。对于基于信赖域的方法，你需要新旧策略间 KL 散度值的指标来告诉你模型是否正常工作。有时你需要输出网络的梯度值来检查它的工作情况。正常网络层的梯度值不应该过大或全为 0，否则它表明要么有异常梯度值，要么是没有梯度流。其他有用的指标有像在输出空间和参数空间的更新步长，对于以上情况，Tensorboard 模块是一个强大的工具，它起初是为 TensorFlow 开发设计的但是后来也支持 PyTorch 框架。它可以简化变量的可视化过程、神经网络计算图等，对实践中使用这些探针很有帮助。

- **使用多个随机种子并计算平均值来减少随机性。**深度强化学习方法很典型地有不稳定的训练过程，随机种子甚至都会很大地影响学习表现，有 NumPy 的随机种子，以及 TensorFlow 或者 PyTorch 的，还有环境的种子等。在随机化这些种子的时候，作为一种默认设置，所有这些种子都需要被合适地随机化。刚开始，你可以固定这些种子，然后观察采样轨迹上是否有任何差异，如果仍有随机性，可能表现系统内还有其他随机因素。固定随机种子可以用来再现学习过程。使用随机种子并得到学习曲线的平均值，可以减少实验对比中深度强化学习随机性造成的得到错误结论的可能性。通常使用越多的随机种子，实验结果就越可靠，但同时也增加了实验耗时。根据经验，我们采用不同的随机种子进行 3 到 5 次试验便可以得到一个相对可信的结果，但是越多越好。
- **平衡 CPU 和 GPU 计算资源以加速训练。**这个提示实际上是关于找到和解决训练速度上的瓶颈问题。在有限的计算机上更好地使用计算资源，对于强化学习要比监督学习复杂。在监督学习中，CPU 经常用于数据读写和预处理，而 GPU 用于进行前向推理和反向传播过程。然而，由于强化学习中推理过程总是涉及与环境的交互，计算梯度的设备需要与处理环境交互的设备匹配计算能力，否则会是对探索或利用的浪费。在强化学习中，CPU 经常被用于与环境交互采样的过程，而这对某些复杂的模拟系统可能涉及大量运算。GPU 被用来进行前向推理和反向传播来更新网络。你在部署大规模训练的过程中，应当检查 CPU 和 GPU 计算资源的利用率，避免线程或进程沉睡。这对于将程序分配到大规模并行计算系统中尤为重要。对于 GPU 过度利用的情况，可以采用更多的采样线程或进程来与环境交互。对于 CPU 过度利用的情况，你可以减少分布式采样线程的数量，或者增加分布式更新线程的数量，增大算法内更新迭代次数，对于离线更新增大批尺寸等，这些都依赖于你管理并行线程和进程的方式。注意上面所述只是关于如何最大化利用你的计算资源，你也应当考虑探索和利用之间的取舍，以及对于多种多样的强化学习任务在不同层次上的采样效率等。为了解决 CPU 和 GPU 资源间的平衡问题，你经常需要在采样和训练过程中使用多线程或多进程并行计算，来充分利用你的计算机。需要仔细考虑如何设计能够同时运行采样线程/进程和网络更新线程/进程的并行训练框架。锁和管道被经常用于这种框架来支持其顺利运行。创建冗余进程有时可以节省等待时间。在线策略和离线策略的处理可能不同，相比于在线策略，离线策略训练的并行设置经常更加灵活，因为你可以在任何时刻更新策略而

非仅在片段的最后一步。一个典型的在 PyTorch 框架下使用多 GPU 分布式训练的使用方式如图 18.3 所示。使用 PyTorch 处理多 GPU 过程在前向推理中采用了一个模型复制过程和一个推理结果采集过程，而在反向更新过程中，梯度缩减被用于并行的梯度反向传播。更多相关细节在强化学习应用的章节中有所讨论，我们也在代码库中提供了一些示例程序。

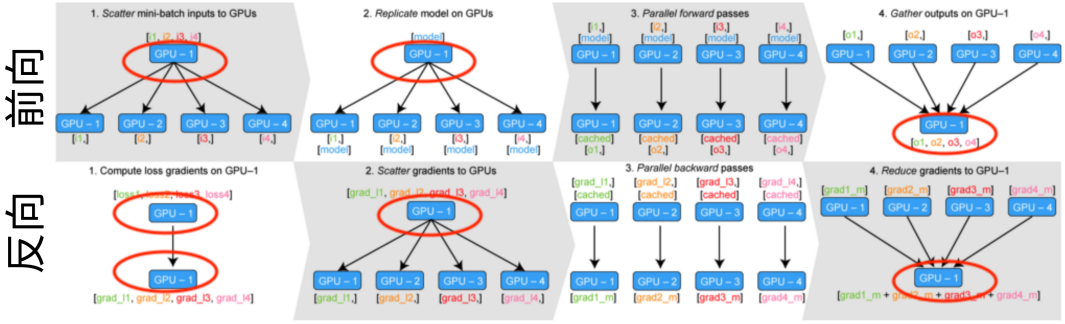

图 18.3　使用 torch.nn.DataParallel 的前向和反向过程（见彩插）

- **可视化**。如果你不能直接从数值中看清潜在关系，你应当尽可能对其可视化。比如，有时由于强化学习过程不稳定的特性，奖励函数可能有很大抖动，这种情况下你可能需要画出奖励值的滑动平均曲线来了解智能体在训练中是否有进步。
- **平滑学习曲线**。强化学习的过程可能非常不稳定。直接从未经处理的学习曲线中得出结论经常是不可靠的，像图 18.4 中未经平滑的学习曲线那样。我们通常要用滑动平均、卷积核等来平滑学习曲线，并且选用一个合适的窗口长度。通过这种方式，学习表现的上升/下降趋势可以更清楚地展示出来，当你在解决一个有着很长训练周期和较慢表现进步的复杂强化学习任务时，这么做可能很关键。

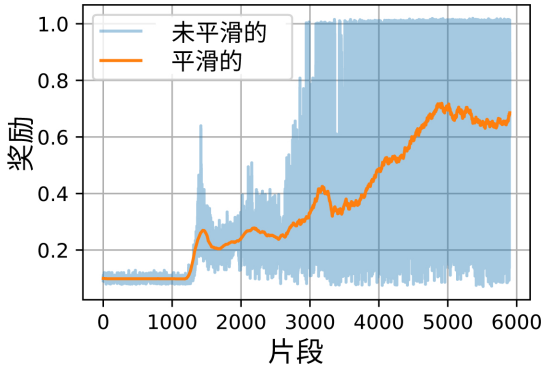

图 18.4　强化学习中平滑的和未平滑的学习曲线（见彩插）

- **理解探索和利用。**从图 18.4 中我们可以看出，学习曲线在早期训练阶段有一个平台期。实际上，这在强化学习过程中不是一个罕见的，而是一个十分常见的情况。这是因为在强化学习中，学习样本不是像在监督学习中那样提前准备好的，而是通过所应用的智能体策略探索得到的。因此，当前策略能否探索到较高奖励值的轨迹在强化学习训练中可能是很关键的。而这会引出探索相关的问题，即需要保证我们的策略能逐渐探索到接近最优的轨迹。当强化学习算法不能对一个具体任务工作时，你需要研究这个智能体是否已经探索到那些更好的轨迹。如果没有，至少说明当前的探索方式可能有问题。然而，如果当前策略能够探索到好的轨迹，但它仍不能收敛到好的动作选择，那么可能是利用问题。这意味着策略不能够较好地从好的轨迹中学习。利用问题可能是由较低的样本效率、较差的价值函数拟合、价值函数较低的学习率、较差的策略网络学习效果等造成的。图 18.4 中所示的学习曲线展示了一个健康的学习进步：一旦好的样本被探索到（在平台期中），策略更新会使学习表现会显著提升（在平台期后）。
- **首先质疑你的算法实现。**当你刚完成代码实现以后，它不会工作，是很常见的，而这时，耐心地调试代码就很重要。算法实现的正确性总是要先于微调一个相对好的结果，因此，应当在保证实现正确性的前提下再考虑微调超参数。而这也正是本章开头提到的强化学习应用的过程：先用小规模例子测试来保证算法实现的正确性，然后逐步扩展到大规模环境并微调分布式的训练过程。一个糟糕的学习表现可能由很多因素导致，如不充足的训练时间、对超参数的糟糕选择、未经归一化的输入数据等，而最常见的原因是代码实现中的错误。

为了给读者提供更全面的关于强化学习算法在具体项目应用上的指导，我们在写本章的过程中也参考了一些外部资源，包括 OpenAI Spinning Up[1]、John Schulman 的幻灯片讲稿[2]、William Falcon 的相关博客[3]等。我们也建议读者参考这些总结得较好的建议和来自研究人员的经验，来帮助实现自己的强化学习算法和应用。查阅与你所做内容相似的他人之前的工作，并从中吸取经验，总是很有帮助的。

此外，读者需要知道只是阅读上面段落中经验性的指示而不实践，几乎没有用。所以，我们强烈推荐读者自己手动实现一些代码来获取实践经验，只用通过这种方式才能发挥这些技巧的最大作用。

参考文献

ESPEHOLT L, SOYER H, MUNOS R, et al., 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures[J]. arXiv preprint arXiv:1802.01561.

[1]OpenAI Spinning Up: `https://spinningup.openai.com/en/latest/index.html`

[2]The Nuts and Bolts of Deep RL Research. John Schulman: `http://joschu.net/docs/nuts-and-bolts.pdf`

[3]Deep RL Hacks: `https://github.com/williamFalcon/DeepRLHacks`

FU J, SINGH A, GHOSH D, et al., 2018. Variational inverse control with events: A general framework for data-driven reward definition[C]//Advances in Neural Information Processing Systems. 8538-8547.

GLOROT X, BENGIO Y, 2010. Understanding the difficulty of training deep feedforward neural networks[C]//Proceedings of the thirteenth international conference on artificial intelligence and statistics. 249-256.

HE K, ZHANG X, REN S, et al., 2016. Deep Residual Learning for Image Recognition[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

HEESS N, HUNT J J, LILLICRAP T P, et al., 2015. Memory-based control with recurrent neural networks[J]. arXiv preprint arXiv:1512.04455.

JADERBERG M, SIMONYAN K, ZISSERMAN A, et al., 2015. Spatial transformer networks[C]// Proceedings of the Neural Information Processing Systems (Advances in Neural Information Processing Systems) Conference. 2017-2025.

MAHMOOD A R, KORENKEVYCH D, VASAN G, et al., 2018. Benchmarking reinforcement learning algorithms on real-world robots[J]. arXiv preprint arXiv:1809.07731.

MNIH V, KAVUKCUOGLU K, SILVER D, et al., 2015. Human-level control through deep reinforcement learning[J]. Nature.

SAXE A M, MCCLELLAND J L, GANGULI S, 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks[J]. arXiv preprint arXiv:1312.6120.

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