强化学习经典文章研读 (I)

贾晖　马燕琳　葛涵　于梅灵　谢君

李庆生　陈子涵　王文远　丁依宁　原增昀

贾顺　徐阳阳　杨以钦　倪元华

倪元华课题组 · 南开大学人工智能学院

2026 年 3 月

前 言

“Reinforcement Learning is Direct Adaptive Optimal Control”

————R. S. Sutton, A. G. Barto, R. J. Williams[1]

IEEE Control Systems Magazine, 1992

强化学习是人工智能领域中实现自主决策与动态优化的核心范式，与监督学习、无监督学习并称为机器学习三大主流技术框架。如今，强化学习已深度渗透机器人控制、智能博弈、工业优化、自动驾驶等诸多领域，推动着人工智能从感知智能向认知智能、决策智能跨越，其理论体系的完善与算法技术的创新，对人工智能领域的发展具有至关重要的意义。南开大学人工智能学院倪元华课题组长期深耕控制科学与人工智能领域的前沿交叉研究与人才培养，为帮助团队成员系统性夯实强化学习理论基础、精准把握领域发展脉络、提升论文研读与科研创新能力，特组织强化学习经典文章研读专题讨论班，以国际经典顶刊顶会论文为核心研读载体，开展体系化、深度化的研读与交流工作。

本讨论班的研读素材均选自 OpenAI Spinning Up 平台发布的经典深度强化学习论文清单（`https://spinningup.openai.com/en/latest/spinningup/keypapers.html`），该清单覆盖无模型强化学习、策略梯度、探索机制、迁移与多任务学习、基于模型的强化学习、元强化学习、模仿学习、安全强化学习等十三大核心方向，收录了领域内具有里程碑意义的经典成果，是强化学习领域入门与深耕的优质参考资料。讨论班秉持“夯实基础、聚焦经典、研思结合”的原则，采用“一人一文、译讲结合、集体研讨”的研读模式，由团队成员各自主讲一篇经典论文。研读过程中，主讲人首先完成论文的精准中英互译，确保对原文核心观点、技术细节的准确理解；随后梳理论文的研究背景、问题提出、核心方法、算法设计、实验验证与结论展望，形成系统的研读框架；最后通过现场讲解、集体提问、深度研讨的形式，与团队成员共同拆解论文中的关键技术点、厘清算法背后的理论逻辑、补全推导细节与思路脉络，在交流与推敲中深化对经典成果的理解与思考。

本笔记是强化学习经典文章研读讨论班全体成员共同努力的结果。从论文的筛选与确定、原文的翻译与校对，到讲解内容的梳理与打磨、课堂研讨的记录与总结，再到笔记的整编、修改与反复校核，每一个环节都经过团队成员的深入交流、细致推敲与严格把控。笔记以各篇经典论文的研读内容为核心主体，在严格

[1]Sutton 和 Barto 是 2024 年图灵奖得主

遵循原文核心框架、符号体系与关键结论的基础上，补充了讲解过程中的拓展分析、问题解答与推导细节，同时融入了团队成员在研读与研讨中的思考与见解，力求内容准确、逻辑清晰、层次分明，既完整还原经典论文的研究精髓与技术内核，又能提供更易理解、更具参考性的研读视角，以期为本课题组后继成员提供一份实用的参考资料。此外，本讨论班的顺利开展与良好氛围，同样离不开所有参与听讲同学的积极投入与热情支持。感谢郭玲利老师，贾茹茹、刘姿含、刘震、谭皓天、吴映桥、方鑫、张晨曦、李佳明、袁梓凌、高浚贺等同学在研讨过程中的认真聆听、积极思考与有益互动，他们的参与为讨论班带来了活跃的交流氛围与宝贵的思想碰撞，也为本笔记的最终完善提供了有力支持。

强化学习领域发展日新月异，新理论、新算法、新应用不断涌现，其理论体系与技术框架仍在持续完善与迭代。由于我们的研究水平与认知深度有限，在论文翻译、内容梳理、笔记整编的过程中，难免存在疏漏、偏颇、表述欠妥或理解不深之处，恳请批评指正。

本笔记仅用于学习交流，不涉及任何商业牟利行为。

倪元华课题组 · 南开大学人工智能学院

2026 年 3 月

作者简介

贾晖　2021 年毕业于河北工业大学，获计算机科学与技术工学学士学位，现为南开大学人工智能学院控制科学与工程专业博士研究生。研究兴趣为分散式最优控制与学习。邮箱：huijia@mail.nankai.edu.cn

马燕琳　2017 年毕业于北京林业大学，获数学与应用数学专业学士学位，现为南开大学人工智能学院人工智能专业博士研究生。研究兴趣为多模态大模型推理、跨模态信息融合与强化学习优化。邮箱：mayanlin19940925@gmail.com

葛涵　2022 年毕业于南开大学，获智能科学与技术工学学士学位，现为南开大学人工智能学院人工智能专业博士研究生。研究兴趣为最优控制、强化学习、智能核聚变。邮箱：1120250307@mail.nankai.edu.cn

于梅灵　2023 年毕业于南开大学，获智能科学与技术工学学士学位，现为南开大学人工智能学院电子信息专业博士研究生。研究兴趣为强化学习、逆最优控制。邮箱：yml@mail.nankai.edu.cn

谢君　2023 年毕业于东南大学，获信息与计算科学理学学士学位，现为南开大学人工智能学院运筹学与控制论专业博士研究生。研究兴趣为数据驱动控制、强化学习、最优控制。邮箱：xiejun@mail.nankai.edu.cn

李庆生　2023 年毕业于大连理工大学，获信息与计算科学学士学位，现为南开大学人工智能学院运筹学与控制论专业博士研究生。研究兴趣为括最优控制、博弈论。邮箱：2120230488@mail.nankai.edu.cn

陈子涵　2024 年毕业于南开大学数学与应用数学专业，获理学学士学位，现为南开大学人工智能学院运筹学与控制论博士研究生，研究兴趣为最优控制。邮箱：1120240253@mail.nankai.edu.cn

王文远　2025 年毕业于南开大学智能科学与技术专业，获工学学士学位，现为南

开大学卓越工程师学院人工智能专业的博士研究生，研究兴趣为强化学习。邮箱：1120250338@mail.nankai.edu.cn

丁依宁 2024 年毕业于东南大学，获信息与计算科学理学学士学位，现为南开大学人工智能学院运筹学与控制论专业硕士研究生。研究兴趣为路径规划、最优控制。邮箱：13068770331@163.com

原增昀 2024 年毕业于中国矿业大学（北京），获信息与计算科学理学学士学位，现为南开大学人工智能学院人工智能专业的硕士研究生。研究兴趣为逆博弈。邮箱：yuanzengyun@mail.nankai.edu.cn

贾顺 2024 年毕业于东北大学，获自动化工学学士学位，现为南开大学人工智能学院人工智能专业硕士研究生。研究兴趣为智能博弈。邮箱：2120240627@mail.nankai.edu.cn

徐阳阳 2024 年毕业于太原科技大学，获自动化工学学士学位，现为南开大学人工智能学院控制工程专业硕士研究生。研究兴趣为安全控制与学习。邮箱：2831830172@qq.com

杨以钦 2024 年于清华大学获控制科学与工程博士学位，现为中国科学院自动化研究所助理研究员。在 NIPS、ICML、ICLR、AAAI、IJCAI、IEEE TASE、IEEE RAL 等期刊与会议上发表论文二十余篇，研究兴趣为智能博弈、强化学习、具身智能。邮箱：yangyiqi19@mails.tsinghua.edu.cn

倪元华 2010 年于中国科学院获运筹学与控制论博士学位，现为南开大学人工智能学院教授、博士生导师，并曾于 2014 年 4 月至 2015 年 5 月在美国加州大学圣地亚哥分校、2016 年 1 月至 2017 年 1 月在香港理工大学担任访问学者。在控制科学方向上的顶级期刊 Automatica、IEEE TAC、SICON 上发表论文 16 篇 (其中短文 2 篇)，研究方向为最优控制、强化学习、智能博弈、最优化理论与应用等。倪元华获第 22 届中国控制会议“关肇直奖”，指导学生论文入选 2024 年亚洲控制会议“Best Student Paper Finalist”，担任国际期刊刊 System & Control Letters、IEEE Control Systems Letters 的编委。邮箱：yhni@nankai.edu.cn

目录

使用深度强化学习玩雅达利（Atari）游戏[1,2]

摘要

本文提出了首个深度学习模型，该模型能够利用强化学习直接从高维感官输入中成功学习控制策略。该模型是一种卷积神经网络，通过 Q-learning 的一个变体进行训练，其输入为原始像素，输出为一个用于估计未来奖励的值函数。本文将该方法应用于街机学习环境（Arcade Learning Environment）中的 7 款雅达利 2600（Atari 2600）游戏，且未对模型架构或学习算法进行任何调整。结果表明，在其中 6 款游戏上，该方法的性能优于此前所有方法；在 3 款游戏上，其性能甚至超过了人类专家。

1 引言

直接从高维感知输入（如视觉和语音）中学习控制智能体，是强化学习（RL）长期以来的挑战之一。大多数在这些领域取得成功的 RL 应用，通常依赖于人工设计的特征，再结合线性价值函数或策略表示。显然，这类系统的性能严重依赖于特征表示的质量。

深度学习的最新进展使得可以从原始感知数据中提取高层特征，从而在计算机视觉[11,16,22] 和语音识别[6,7] 中取得了突破。这些方法利用了多种神经网络结构，包括卷积网络、多层感知机、受限玻尔兹曼机以及递归神经网络，并结合了有监督与无监督学习。自然地会提出这样的问题：类似的技术是否也能对处理感知数据的强化学习有益？

然而，从深度学习的角度来看，强化学习面临若干挑战。首先，迄今为止大多数成功的深度学习应用都需要大量的人工标注训练数据。相比之下，RL 算法必须能够仅从一个标量奖励信号中学习，而该信号往往是稀疏的、噪声较大的，并且存在延迟。动作与结果奖励之间的延迟可能长达上千步，与监督学习中输入和目标之间的直接对应相比，这显得尤其艰难。另一个问题是，大多数深度学习算法假设数据样本相互独立，而在 RL 中通常遇到的是高度相关的状态序列。此外，在 RL 中，数据分布会随着算法学习到新的行为而不断变化，这对假设底层分布固定

[1] 原文：Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning [J]. arXiv preprint arXiv:1312.5602, 2013.

[2] 译者：贾晖。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

的深度学习方法而言是个难题。

本文表明，卷积神经网络能够克服这些挑战，在复杂的 RL 环境中从原始视频数据中学习成功的控制策略。该网络使用 Q-learning[26] 的一种变体训练，并通过随机梯度下降来更新权重。为缓解数据相关性和非平稳分布的问题，本文使用了一种经验回放机制 (*experience reply mechanism*)[13]，它通过随机抽取过去的转移，从而在大量过去行为上平滑训练分布。

本文将方法应用于 Arcade Learning Environment (ALE)[3] 中实现的一系列 Atari 2600 游戏。Atari 2600 是一个具有挑战性的 RL 测试平台，它向智能体提供高维视觉输入（210×160 的 RGB 视频，60Hz），并包含一系列丰富多样且对人类玩家也很有难度的任务。本文的目标是构建一个单一的神经网络智能体，能够尽可能多地学会玩这些游戏。网络没有被提供任何与具体游戏相关的信息或手工设计的视觉特征，也无法访问模拟器的内部状态；它只能从视频输入、奖励与终止信号以及动作集合中学习，就像人类玩家一样。此外，网络结构与所有训练所用的超参数在不同游戏间保持完全一致。截至目前，该网络在七个游戏中的六个上超越了所有先前的 RL 算法，并且在其中三个游戏中超过了专家人类玩家。图 1 展示了训练所用的五个游戏的截图样例。

图 0.1　来自五个 Atari 2600 游戏的屏幕截图：（从左到右）Pong、Breakout、Space Invaders、Seaquest、Beam Rider。

2 背景

本文考虑这样一类任务：一个智能体以一系列动作、观测和奖励的形式与环境 $\mathcal{E}$（在本文中即 Atari 模拟器）交互。在每个时间步，智能体从合法游戏动作集合 $\mathcal{A} = \{1, \ldots, K\}$ 中选择一个动作 a_t。该动作被传递给模拟器，从而修改其内部状态以及游戏得分。一般来说，$\mathcal{E}$ 可能是随机的。模拟器的内部状态对智能体是不可见的；相反，它接收到的是一幅图像 $x_t \in \mathbb{R}^d$，该图像由原始像素值组成，表示当前屏幕。此外，它还会接收到一个奖励 r_t，代表游戏得分的变化。需要注意的是，一般情况下，游戏得分可能依赖于之前的整条动作与观测序列；某个动作的反馈可能要在经过上千个时间步后才能收到。

由于智能体只能观测当前屏幕的图像，因此该任务是部分可观测的，并且许

多模拟器状态在感知上是不可区分的，即仅凭当前屏幕 x_t 无法完全理解当前局势。因此，本文考虑由动作和观测组成的序列：

$$s_t = (x_1, a_1, x_2, \ldots, a_{t-1}, x_t),$$

并学习依赖于这些序列的游戏策略。假设模拟器中的所有序列都在有限的时间步内终止。这一形式化方法生成了一个庞大但有限的马尔可夫决策过程（MDP），其中每条序列都是一个不同的状态。因此，本文可以通过将完整序列 s_t 作为时刻 t 的状态表示，来直接应用标准的 MDP 强化学习方法。

智能体的目标是在与模拟器交互时选择动作，以最大化未来的奖励。本文做出标准假设：未来的奖励在每个时间步的折扣因子为 γ，并定义时刻 t 的未来折扣回报为

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'},$$

其中 T 是游戏终止的时间步。本文定义最优动作-价值函数 $Q^*(s,a)$ 为：在看到某个序列 s 并执行动作 a 后，遵循任意策略所能达到的最大期望回报：

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\big[R_t \,\big|\, s_t = s, a_t = a, \pi\big],$$

其中 π 是一个将序列映射到动作（或动作分布）的策略。

最优动作-价值函数满足一个重要的恒等式，称为 Bellman 方程。其直观解释如下：如果已知在下一时间步某个序列 s' 的最优价值 $Q^*(s',a')$，对所有可能的动作 a' 都已知，那么最优策略就是选择能够最大化 $r+\gamma Q^*(s',a')$ 的期望值的动作 a'：

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\Big[r + \gamma \max_{a'} Q^*(s',a') \,\Big|\, s,a\Big]. \tag{1}$$

许多强化学习算法的基本思想，是通过使用 Bellman 方程作为迭代更新来估计动作-价值函数：

$$Q_{i+1}(s,a) = \mathbb{E}\Big[r + \gamma \max_{a'} Q_i(s',a') \,\Big|\, s,a\Big].$$

这样的值迭代算法会收敛到最优动作-价值函数，即 $Q_i \to Q^*$ 当 $i \to \infty$[23]。然而在实践中，这种基本方法完全不可行，因为它需要分别为每一个序列估计动作-价值函数，无法进行任何泛化。因此，通常会使用函数逼近器来估计动作-价值函数：

$$Q(s,a;\theta) \approx Q^*(s,a).$$

在强化学习领域，这通常是一个线性函数逼近器，但有时也使用非线性函数逼近器，例如神经网络。本文称带参数 θ 的神经网络函数逼近器为 Q-网络。Q-网络可

以通过最小化一系列随迭代 i 变化的损失函数 $L_i(\theta_i)$ 来训练：

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\Big[\big(y_i - Q(s,a;\theta_i)\big)^2\Big], \tag{2}$$

其中

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\Big[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \,\Big|\, s,a\Big]$$

是第 i 次迭代的目标，而 $\rho(s,a)$ 是序列 s 与动作 a 上的概率分布，本文称之为行为分布。在优化损失函数 $L_i(\theta_i)$ 时，固定前一次迭代的参数 θ_{i-1}。需要注意的是，目标依赖于网络权重；这与监督学习中在学习开始前就固定目标不同。对损失函数关于权重求导，得到以下梯度：

$$\begin{aligned}\nabla_{\theta_i} L_i(\theta_i) &= \mathbb{E}_{s,a\sim\rho(\cdot)}\big[\nabla_{\theta_i}(y_i - Q(s,a;\theta_i))^2\big] \\ &= \mathbb{E}_{s,a\sim\rho(\cdot)}\big[-2\,(y_i - Q)\,\nabla_{\theta_i} Q(s,a;\theta_i)\big] \\ &= -2\mathbb{E}_{s,a\sim\rho(\cdot);\,s'\sim\mathcal{E}}\Big[\big(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\big)\nabla_{\theta_i} Q(s,a;\theta_i)\Big].\end{aligned} \tag{3}$$

在实际中，与其在上述梯度中计算完整期望，不如采用随机梯度下降优化损失函数，这在计算上更加高效。如果在每个时间步之后更新权重，并将期望替换为分别来自行为分布 ρ 与模拟器 $\mathcal{E}$ 的单一样本，那么就得到熟知的 Q-learning 算法[26]:

$$\begin{aligned}\theta_{i+1} &= \theta_i - \alpha\nabla_{\theta_i} L_i(\theta_i) \\ &= \theta_i + 2\alpha\Big[\big(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\big)\nabla_{\theta_i} Q(s,a;\theta_i)\Big],\end{aligned}$$

即和环境交互一条轨迹，得到一个样本 (s,a,r,s'), 用这个样本近似期望：$y = r + \gamma \max_{a'} Q(s',a';\theta^-)$ 这里 θ^- 是固定的旧参数。更新当前 Q 网络，让 $Q(s,a;\theta)$ 更接近目标 y。

需要注意的是，该算法是无模型 *(model-free)* 的：它直接使用来自模拟器 $\mathcal{E}$ 的样本来解决强化学习问题，而无需显式构建 $\mathcal{E}$ 的估计。同时它也是 *off-policy* 的：它学习的是贪婪策略

$$a = \arg\max_a Q(s,a;\theta),$$

而在训练过程中遵循的行为分布能够保证对状态空间进行充分探索。在实践中，行为分布通常通过 ε-贪婪策略选择，即以概率 $1-\varepsilon$ 选择贪婪动作，以概率 ε 随机选择动作。

3 相关工作

也许强化学习中最著名的成功案例是 *TD-Gammon*，这是一个西洋双陆棋程序，它完全通过强化学习和自我对弈学习，最终达到了超越人类的水平[24]。TD-Gammon 使用了一种类似于 Q-learning 的无模型强化学习算法，并通过一个只有一层隐藏层的多层感知机来逼近价值函数。[2]

然而，随后尝试将该方法应用到国际象棋、围棋和跳棋等游戏时，并未取得类似的成功。这导致一种广泛的看法：TD-Gammon 的方法只是一个特例，它仅在西洋双陆棋中有效，原因可能是骰子的随机性有助于探索状态空间，并且使得价值函数特别平滑[19]。

此外，已有研究表明，将无模型强化学习算法（如 Q-learning）与非线性函数逼近器[25] 结合，或者与 *off-policy* 学习[1] 结合，都可能导致 Q-网络发散。随后，大多数强化学习的研究都集中在收敛性更好保证的线性函数逼近器上[25]。

最近，结合深度学习与强化学习的研究重新引起了广泛兴趣。深度神经网络被用于估计环境 $\mathcal{E}$；受限玻尔兹曼机被用于估计价值函数[21] 或策略[9]。此外，Q-learning 的发散问题也部分通过梯度时序差分方法得到了解决。这些方法已经被证明在使用非线性函数逼近器来评估固定策略时是收敛的[14]；或者在使用线性函数逼近器并结合 Q-learning 的一个受限变体来学习控制策略时是收敛的[15]。然而，这些方法尚未扩展到非线性控制。

与本文方法最相似的已有工作可能是 *神经拟合 Q-learning*（*NFQ*）[20]。NFQ 优化式 (2) 中所示的一系列损失函数，并使用 RPROP 算法更新 Q-网络的参数。然而，它采用批量更新方式，其每次迭代的计算成本与数据集规模成正比；相比之下，本文的方法使用随机梯度更新，其每次迭代的成本为低常数，并能扩展到大规模数据集。NFQ 也已成功应用于一些简单的实际控制任务，输入完全是视觉数据。该方法首先使用深度自编码器学习任务的低维表示，然后在该表示上应用 NFQ[12]。与之不同，本文的方法是端到端的强化学习，直接从视觉输入出发；因此它可以学习到与动作-价值判别直接相关的特征。Q-learning 先前也曾与经验回放和一个简单的神经网络结合使用[13]，但那时的输入是低维状态而不是原始视觉输入。

将 Atari 2600 模拟器作为强化学习平台的做法最早由[3] 提出，他们使用带有线性函数逼近和通用视觉特征的标准强化学习算法。随后，研究者通过增加更多特征，并使用 *tug-of-war hashing* 技术 (一种随机特征压缩方法，属于传统特征工程手段，而不是端到端学习) 将特征随机投影到低维空间，从而提升了效果[2]。进化计算框架 *HyperNEAT*[8] 也被应用到 Atari 平台，在该方法中为每个不同的游戏分别进

[2]事实上，TD-Gammon 逼近的是状态价值函数 $V(s)$ 而不是动作-价值函数 $Q(s,a)$，并且它直接通过自我对弈的样本进行在策略学习。

化出一个神经网络来表示游戏策略。当反复在确定性序列下训练（利用模拟器的重置功能）时，这些策略能够利用若干 Atari 游戏中的设计缺陷。

4 深度强化学习

计算机视觉和语音识别中的最新突破依赖于在非常大的训练集上高效训练深度神经网络。最成功的方法通常直接从原始输入进行训练，并使用基于随机梯度下降的轻量更新。通过向深度神经网络输入足够的数据，往往可以学到比人工设计特征更好的表示[11]。这些成功案例为本文的强化学习方法提供了启发。本文的目标是将强化学习算法与深度神经网络相结合，使其能够直接作用于 RGB 图像，并通过使用随机梯度更新高效处理训练数据。

Tesauro 的 TD-Gammon 架构为这种方法提供了一个起点。该架构通过直接利用来自环境交互（或在西洋双陆棋的情况下，来自自我对弈）的在策略经验样本 $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ 来更新估计价值函数的网络参数。由于这种方法在 20 年前就已经超越了人类最强的西洋双陆棋玩家，因此自然会想到：如果结合二十年来硬件的改进、现代深度神经网络架构以及可扩展的 RL 算法，是否能够取得显著进展。

与 TD-Gammon 和类似的在线方法相反，本文采用了一种称为*经验回放 (experience replay)* 的技术[13]。在该技术中，在每个时间步存储智能体的经验 $e_t = (s_t, a_t, r_t, s_{t+1})$，并将其汇集到一个回放存储器 $D = \{e_1, \ldots, e_N\}$ 中，包含多个回合的经验。在算法的内循环中，本文对从存储池中随机抽取的经验样本 $e \sim D$ 进行 Q-learning 更新或小批量更新。在完成经验回放后，智能体根据一个 ε-贪婪策略选择并执行动作。由于将任意长度的历史作为神经网络输入可能比较困难，本文的 Q 函数改为使用由函数 ϕ 生成的固定长度历史表示。完整的算法称为 *深度 Q-learning*，如算法 1 所示。

这种方法相比标准的在线 Q-learning[23] 有若干优势。首先，每一步经验都可能在多次权重更新中被使用，这提高了数据效率。其次，直接从连续样本中学习效率很低，因为样本之间存在强相关性；通过对样本进行随机化，可以打破这些相关性，从而减少更新的方差。第三，在 on-policy 学习中，当前参数决定了下一个训练样本。例如，如果当前的最优动作是“向左移动”，那么训练样本将主要来自左侧；而当最优动作切换为“向右移动”时，训练分布也会随之切换。这很容易导致不希望得到的反馈回路，使得参数陷入较差的局部最优，甚至灾难性地发散[25]。通过使用经验回放，行为分布被平均在其许多历史状态上，从而平滑了学习过程，避免了参数的震荡或发散。需要注意的是，在经验回放下学习时，必须采用 off-policy 学习（因为当前参数与生成样本时的参数不同），这也正是本文选择 Q-learning 的动机。

在实际中，本文的算法仅在回放存储器中存储最近的 N 条经验，并在更新时从 D 中均匀随机采样。这种方法在某些方面受到限制，因为存储缓冲区并未区分重要转移，而且由于内存容量有限，总是会用最新的转移覆盖旧的转移。同样地，均匀采样会给予所有转移相同的重要性。一个更复杂的采样策略可能会强调那些最能带来学习收益的转移，类似于 优先扫掠（*prioritized sweeping*）[17]。

算法 1 带经验回放的深度 Q 学习 (Deep Q-learning with Experience Replay)

输入: 回放记忆容量 N，折扣因子 γ，探索概率 ε
输出: 动作-价值函数 Q 的参数 θ

初始化回放记忆库 D，容量为 N
随机初始化动作-价值函数 Q 的参数 θ
for 每一回合 $= 1, \ldots, M$ **do**
    初始化初始序列 $s_1 = \{x_1\}$，并进行预处理得到 $\phi_1 = \phi(s_1)$
    for 每个时间步 $t = 1, \ldots, T$ **do**
        以概率 ε 选择一个随机动作 a_t
        否则选择 $a_t = \arg\max_a Q^*(\phi(s_t), a; \theta)$
        在模拟器中执行动作 a_t，观察奖励 r_t 和下一帧图像 x_{t+1}
        更新序列 $s_{t+1} = (s_t, a_t, x_{t+1})$，并进行预处理 $\phi_{t+1} = \phi(s_{t+1})$
        将转移样本 $(\phi_t, a_t, r_t, \phi_{t+1})$ 存入记忆库 D
        从 D 中随机采样一个小批量转移样本 $(\phi_j, a_j, r_j, \phi_{j+1})$
        对每个样本，设定目标值：

$$y_j = \begin{cases} r_j & \text{若 } \phi_{j+1} \text{ 为终止状态} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{若 } \phi_{j+1} \text{ 为非终止状态} \end{cases}$$

        根据公式 (3)，对损失函数

$$(y_j - Q(\phi_j, a_j; \theta))^2$$

        执行一次梯度下降更新
    end for
end for

注 1：需要注意的是，如果直接按照式 $(y_j - Q(\phi_j, a_j; \theta))^2$ 对参数执行梯度下降，会面临较大困难。其原因在于目标项

$$y_i = r + \gamma \max_{a'} Q(s', a'; \theta)$$

本身依赖于参数 θ，而预测项 $Q(s, a; \theta)$ 同样依赖于 θ。这样一来，梯度不仅会通过预测项传播，还会通过目标项传播；再加上 max 运算在动作空间上的非光滑性，使得整体梯度几乎无法稳定计算。

为了解决这一问题，后续方法普遍引入了两个网络参数集：主网络 w 与目标

网络 w_T。其中，目标网络 w_T 用于计算相对稳定的目标值：

$$y_T = r + \gamma \max_{a'} \hat{Q}(s', a'; w_T),$$

而主网络 w 则用于计算当前的动作价值预测 $\hat{Q}(s, a; w)$，并通过最小化平方误差

$$(y_T - \hat{Q}(s, a; w))^2$$

来更新参数。由于目标网络 w_T 在训练中保持固定，仅在每隔 C 次迭代后才从主网络复制参数（即 $w_T \leftarrow w$），因此目标项在一段时间内相对稳定，有效避免了梯度的不确定传播和训练的不稳定性。

这种双网络机制是深度 Q 网络（DQN）的核心改进之一，它将原本不可行的直接梯度下降转化为可计算、可稳定优化的形式，从而使深度强化学习在高维状态空间上得以应用。

注 2(小批量更新)： 从回放池 D 中均匀采样一个小批量 $\mathcal{B} = \{(\phi_j, a_j, r_j, \phi'_j, \text{done}_j)\}_{j=1}^{B}$，其中 B 为批量大小（如 $B = 32$）。用目标网络参数 θ^- 计算每个样本的目标值

$$y_j \;=\; r_j \;+\; \gamma\,(1 - \text{done}_j)\,\max_{a'} Q(\phi'_j, a';\, \theta^-).$$

定义小批量的平均平方损失

$$\mathcal{L}(\theta) \;=\; \frac{1}{B}\sum_{j=1}^{B}\Big(y_j - Q(\phi_j, a_j;\, \theta)\Big)^2 \;=\; \frac{1}{B}\sum_{j=1}^{B}\delta_j^2, \qquad \delta_j \;=\; y_j - Q(\phi_j, a_j;\, \theta).$$

对该平均损失做一次梯度下降：

$$\theta \;\leftarrow\; \theta \;-\; \alpha\,\nabla_\theta \mathcal{L}(\theta) \;=\; \theta \;-\; \frac{\alpha}{B}\sum_{j=1}^{B}\Big(-2\,\delta_j\,\nabla_\theta Q(\phi_j, a_j;\, \theta)\Big),$$

常见实现会把常数 2 吸收到学习率里，写成

$$\theta \;\leftarrow\; \theta \;+\; \frac{\alpha}{B}\sum_{j=1}^{B}\delta_j\,\nabla_\theta Q(\phi_j, a_j;\, \theta).$$

为稳定训练，目标网络参数每隔 C 次迭代用主网络参数同步一次：

$$\theta^- \leftarrow \theta \quad （每\,C\,步）。$$

4.1 预处理与模型结构

直接处理原始 Atari 帧（210×160 像素图像，128 色调色板）在计算上开销很大，因此本文应用了一个基本的预处理步骤来降低输入维度。原始帧首先由 RGB 转换为灰度图像，并下采样到 110×84 的尺寸。最终的输入表示通过裁剪一块 84×84 的

区域来获得，该区域大致覆盖了游戏的主要活动区域。最后一步裁剪仅仅是因为本文使用的 GPU 卷积实现[11] 期望输入为方形。在本文实验中，算法 1 中的函数 ϕ 将这种预处理应用于历史的最近 4 帧，并将它们堆叠起来作为 Q 函数的输入。

有多种方式可以用神经网络对 Q 进行参数化。由于 Q 函数会将“历史 - 动作”对映射为其 Q 值的标量估计，因此在以往的部分研究方法中[12,20]，会将历史信息与动作作为神经网络的输入。这类架构的主要缺陷在于，计算每个动作的 Q 值都需要单独执行一次前向传播，导致计算成本随动作数量的增加呈线性增长。相反，本文采用的架构为每个可能的动作设置了一个独立的输出单元，且仅将状态表征作为神经网络的输入。换句话说，神经网络的输出层是一个长度为动作数的向量，其中每个分量对应一个动作的预测 Q 值。这样在给定状态下，只需一次前向传播，就能同时得到所有动作的 Q 值，而不必像状态 + 动作作为输入的方法那样，为每个动作单独运行一次网络，计算效率显著提高。

接下来本文描述在所有七个 Atari 游戏中使用的精确网络结构。神经网络的输入是一个 84×84×4 的图像，由函数 ϕ 生成，它表示最近 4 帧预处理后的游戏画面。第一层隐藏层对输入图像应用 16 个 8×8 卷积核（步长为 4），并使用修正线性单元（ReLU）作为非线性激活函数[10,18]。ReLU 的形式是 $\mathrm{ReLU}(x) = \max(0, x)$，即把负数截断为 0，正数保持不变，它能增加网络的非线性表达能力，同时计算开销小。第二层隐藏层使用 32 个 4×4 卷积核（步长为 2），同样使用 ReLU 激活。最后一层隐藏层是一个全连接层，包含 256 个 ReLU 单元，用来进一步整合卷积特征。输出层是一个全连接的线性层，每个合法动作对应一个输出节点，表示在该状态下该动作的预测 Q 值。本文考虑的游戏中，合法动作的数量在 4 到 18 之间不等。我们将通过该方法训练的卷积网络称为深度 Q 网络（*DQN*）。

5 实验

到目前为止，本文在七个流行的 Atari 游戏上进行了实验——Beam Rider、Breakout、Enduro、Pong、Q*bert、Seaquest、Space Invaders。本文在 7 款游戏中均采用相同的网络架构、学习算法及超参数设置，这表明我们的方法具备足够的鲁棒性，无需融入游戏专属信息，即可在多种游戏上有效运行。尽管我们在未经修改的真实游戏环境中评估智能体性能，但仅在训练阶段对游戏的奖励结构做出了一处调整：由于不同游戏的得分规模差异极大，本文将所有正奖励固定为 1，所有负奖励固定为 -1，0 奖励则保持不变。通过这种方式对奖励进行裁剪，能够限制误差导数的规模，从而更易于在多个游戏中使用相同的学习率。与此同时，这种做法也可能对智能体的性能产生影响——因为智能体无法再区分不同幅度的奖励（例如，原本 $+10$ 和 $+1$ 的奖励会被统一处理为 $+1$，导致智能体无法识别更大

奖励的行为）。

在这些实验中，本文使用了 RMSProp 算法，批量大小为 32。在训练过程中，行为策略采用 ε-贪婪方法，其中 ε 在最初的一百万帧中线性从 1 下降到 0.1，之后固定在 0.1。本文总共训练了一千万帧，并使用了包含最近一百万帧的回放存储器。

参考此前用于玩 Atari 的研究方法，本文也使用了一个简单的 跳帧技术[3]。更具体地说，智能体并非每帧画面都进行观察和动作选择，而是每隔 k 帧才执行一次观察与选动作操作，在被跳过的帧中，智能体则重复执行上一次选择的动作。由于让模拟器向前运行一步（即生成一帧画面）所需的计算量，远小于让智能体选择一次动作的计算量，因此该技术能让智能体在运行时间未显著增加的前提下，大致多玩 k 倍次数的游戏。在所有游戏中，我们均设置 $k = 4$（即每 4 帧观察并选一次动作）；但在 Space Invaders 这款游戏中，我们发现若使用 $k = 4$，会因激光闪烁的周期特性导致激光不可见（注：激光闪烁频率与帧跳过间隔重叠，使得智能体观察到的帧中恰好没有激光画面，无法捕捉激光这一关键环境信息）。因此，我们为该游戏将 k 调整为 3，以确保激光能被智能体观察到，这是不同游戏之间超参数设置的唯一差异。

5.1 训练与稳定性

在监督学习中，可以很容易地通过在训练集和验证集上评估模型来跟踪训练过程中的性能。然而在强化学习中，准确评估智能体在训练过程中的进展是有挑战的。本文采用[3] 建议的评估指标，即智能体在若干回合或游戏中所获得的总奖励的平均值，并定期在训练过程中计算该指标。平均总奖励往往有很大的噪声性，因为策略参数的微小变化可能会导致智能体所访问状态分布的巨大变化。图 2 最左边的两幅图展示了在 Seaquest 和 Breakout 游戏训练过程中平均总奖励的变化。这两个平均奖励曲线确实都存在明显噪声，给人的印象是学习算法并没有稳定进展。另一个更稳定的指标是策略的估计动作-价值函数 Q，它提供了从任一给定状态出发，按照策略能获得的折扣奖励的估计值。本文在训练开始前通过运行随机策略收集一组固定状态，并跟踪这些状态下最大预测 Q 的平均值。图 2 最右边的两幅图展示了在 Breakout 和 Seaquest 上该平均最大预测 Q 的变化情况。相比平均总奖励，这些曲线更加平滑。在其他五个游戏上绘制相同指标也得到类似的平滑曲线。除了在训练过程中观察到预测 Q 的平滑改进外，本文在任何实验中都没有遇到发散问题。这表明，尽管缺乏任何理论上的收敛保证，本文的方法仍能够稳定地通过强化学习信号和随机梯度下降来训练大型神经网络。

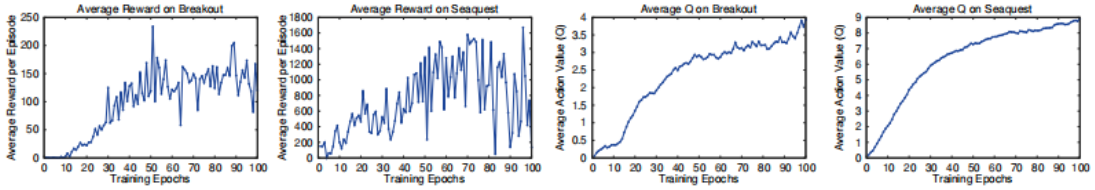


图 0.2　左边两幅图分别展示了 Breakout 和 Seaquest 在训练过程中每回合的平均奖励。该统计量是通过运行一个 ε-贪婪策略（$\varepsilon = 0.05$）并执行 10000 步得到的。右边两幅图分别展示了 Breakout 和 Seaquest 上一组保留状态集合的预测动作价值函数最大值的平均。一个 epoch 对应 50000 次小批量权重更新，约等于 30 分钟的训练时间。

图 0.3　最左边的图展示了 Seaquest 游戏中一段 30 帧序列的预测价值函数。图中的三张截图分别对应 A、B、C 标注的帧。

5.2 价值函数可视化

图 3 展示了在 Seaquest 游戏中学习到的价值函数的可视化。图中显示，当敌人出现在屏幕左侧时（点 A），预测值会跳升。随后，智能体向敌人发射鱼雷，在鱼雷即将击中敌人时，预测值达到峰值（点 B）。最后，当敌人消失时，预测值下降到接近原始水平（点 C）。图 3 表明，本文的方法能够学习出在相对复杂的事件序列下价值函数的动态演化过程。

5.3 主要评估

本文将结果与 RL 文献中表现最好的方法进行比较[3,4]。其中，被标记为 **Sarsa** 的方法使用 **Sarsa** 算法，在 Atari 任务上基于若干人工设计的特征集学习线性策略，本文报告的是表现最好的特征集得分[3]。**Contingency** 方法与 **Sarsa** 类似，但它在特征集中加入了对屏幕中处于智能体控制下区域的学习表征[4]。需要注意的是，这两种方法通过背景减除，并将 128 种颜色分别作为独立通道，从而引入了大量关于视觉问题的先验知识。由于许多 Atari 游戏为每种对象类型使用一种独特颜色，因此将每种颜色作为一个独立通道等价于生成每种对象类型存在与否的独立二值图。在对比之下，本文的智能体只接收原始 RGB 屏幕截图作为输入，必须自己学习如何检测对象。除了学习到的智能体之外，本文还报告了人类专家玩家以及随机策略的得分。人类表现是指经过大约两小时的游戏后获得的奖励中位数。需要注意的是，本文报告的人类分数远高于 Bellemare 等人[3] 中的数值。对于学习方法，本文遵循 Bellemare 等人[3,5] 的评估策略，报告智能体在固定步数下运行 ε-贪婪策

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

表 1: 上半部分表格比较了在固定步数下运行 ε-贪婪策略（$\varepsilon = 0.05$）时，各种学习方法的平均总奖励。下半部分表格报告了 HNeat 和 DQN 的单次最佳回合表现。HNeat 产生的是确定性策略，每次都会得到相同的分数；而 DQN 使用的是 ε-贪婪策略（$\varepsilon = 0.05$）。

略（$\varepsilon = 0.05$）所获得的平均分数。表 1 的前五行展示了所有游戏的每局平均得分。本文的方法（标记为 DQN）在所有七个游戏上都以显著优势超过了其他学习方法，尽管它几乎没有利用关于输入的先验知识。

此外，本文在表 1 的最后三行中加入了与[8] 提出的进化策略搜索方法（一种基于进化算法的神经网络搜索框架）的比较。对于该方法，本文报告了两组结果：**HNeat Best** 得分是通过使用一个人工设计的对象检测器算法（该算法输出 Atari 屏幕上对象的位置和类型）得到的；**HNeat Pixel** 得分则通过使用 Atari 模拟器的特殊 8 色通道表征获得，该表征中每个通道对应一个物体标签映射，即每个通道专门编码某一类物体的分布。需要说明的是，这种进化策略搜索方法严重依赖于找到一个确定性的状态序列，该序列需能实现对游戏的有效利用，指可稳定获得高分的特定操作流程。但通过这种方式学到的策略，很难泛化到随机扰动场景中，面对游戏状态的微小随机变化时，策略易失效。因此，该算法仅在得分最高的单个回合上进行评估。相比之下，本文的方法是在 ε-贪婪控制序列下评估的，因此必须能够在各种可能的情境下进行泛化。尽管如此，本文仍然表明，除 Space Invaders 外，本文的方法在所有游戏上的平均表现（表 1 第 4 行）和最好表现（表 1 第 8 行）都超过了进化策略搜索方法。

最后，本文展示了本文的方法在 Breakout、Enduro 和 Pong 上的表现超过了人类专家，并且在 Beam Rider 上接近人类水平。而在 Q*bert、Seaquest 和 Space Invaders 这些游戏上，本文与人类差距较大，因为这些游戏要求网络找到一种能够在长时间尺度上维持的策略。

6 结论

本文提出了一种新的深度学习模型用于强化学习，并展示了它能够仅凭原始像素输入，就掌握 Atari 2600 游戏中的复杂控制策略。本文还提出了一种在线 Q-learning 的变体，该方法结合了随机小批量更新与经验回放存储，从而简化了深度

网络在强化学习中的训练。本文的方法在七个游戏中的六个上达到了当时最优的结果，且全程未对模型架构或超参数进行任何调整。

参考文献

[1] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In Proceedings of the 12th International Conference on Machine Learning (ICML 1995), pages 30–37. Morgan Kaufmann, 1995.

[2] Marc Bellemare, Joel Veness, and Michael Bowling. Sketch-based linear value function approximation. In Advances in Neural Information Processing Systems 25, pages 2222–2230, 2012.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, 2013.

[4] Marc G Bellemare, Joel Veness, and Michael Bowling. Investigating contingency awareness using atari 2600 games. In AAAI, 2012.

[5] Marc G. Bellemare, Joel Veness, and Michael Bowling. Bayesian learning of recursively factored environments. In Proceedings of the Thirtieth International Conference on Machine Learning (ICML 2013), pages 1211–1219, 2013.

[6] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. Audio, Speech, and Language Processing, IEEE Transactions on, 20(1):30 –42, January 2012.

[7] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In Proc. ICASSP, 2013.

[8] Matthew Hausknecht, Risto Miikkulainen, and Peter Stone. A neuro-evolution approach to general atari game playing. 2013.

[9] Nicolas Heess, David Silver, and Yee Whye Teh. Actor-critic reinforcement learning with energy-based policies. In European Workshop on Reinforcement Learning, page 43, 2012.

[10] Kevin Jarrett, Koray Kavukcuoglu, MarcAurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In Proc. International Conference on Computer Vision and Pattern Recognition (CVPR 2009), pages 2146–2153. IEEE, 2009.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 1106–1114, 2012.

[12] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In Neural Networks (IJCNN), The 2010 International Joint Conference on, pages 1–8. IEEE, 2010.

[13] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

[14] Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Rich Sutton. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In Advances in Neural Information Processing Systems 22, pages 1204–1212, 2009.

[15] Hamid Maei, Csaba Szepesv´ari, Shalabh Bhatnagar, and Richard S. Sutton. Toward off-policy learning control with function approximation. In Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pages 719–726, 2010.

[16] Volodymyr Mnih. Machine Learning for Aerial Image Labeling. PhD thesis, University of Toronto, 2013.

[17] Andrew Moore and Chris Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. Machine Learning, 13:103–130, 1993.

[18] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pages 807–814, 2010.

[19] Jordan B. Pollack and Alan D. Blair. Why did td-gammon work. In Advances in Neural Information Processing Systems 9, pages 10–16, 1996.

[20] Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In Machine Learning: ECML 2005, pages 317–328. Springer, 2005.

[21] Brian Sallans and Geoffrey E. Hinton. Reinforcement learning with factored states and actions. Journal of Machine Learning Research, 5:1063–1088, 2004.

[22] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In Proc. International Conference on Computer Vision and Pattern Recognition (CVPR 2013). IEEE, 2013.

[23] Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.

[24] Gerald Tesauro. Temporal difference learning and td-gammon. Communications of the ACM, 38(3):58–68, 1995.

[25] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Automatic Control, IEEE Transactions on, 42(5):674–690, 1997.

[26] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.

深度循环 Q 学习在部分可观测的马尔可夫决策过程中的研究[1] [2]

摘要

深度强化学习已实现在复杂任务中训练出高效控制器的能力。然而，这些控制器的记忆能力有限，并且在每个决策时刻都依赖于完整感知的游戏画面。为了弥补这些缺陷，本文将深度 Q 网络（DQN）中第一个卷积层后的全连接层替换为循环 LSTM 层，研究了引入循环机制所带来的影响。由此产生的深度循环 Q 网络（DRQN），尽管在每个时间步只能看到单帧图像，却能成功地跨时间整合信息，在标准雅达利游戏以及具有闪烁屏幕、部分可观察特征的改良版游戏中，复现了 DQN 的表现能力。此外，当 DRQN 在部分观测条件下训练，并逐步在更完整的观测条件下进行评估时，其性能会随着可观测性的提高而提升。相反，当 DRQN 在完全观测条件下训练，并在部分观测条件下进行评估时，其性能下降幅度小于 DQN。因此，在提供相同长度历史信息的情况下，使用循环结构是 DQN 输入层堆叠历史帧的一种可行替代方案；尽管在学习游戏策略时，循环结构并不带来系统性优势，但在评估阶段，若观测质量发生变化，循环网络能更好地适应。

1 引言

深度 Q 网络（DQN）已被证明能够在多种雅达利 2600 游戏中学习出达到人类水平的控制策略（Mnih 等，2015）。正如其名，DQN 学习的估计从当前游戏状态出发，计算每个可能动作所对应的 Q 值（即长期折扣回报）。只要网络的 Q 值估计足够准确，就可以通过在每一时间步选择具有最大 Q 值的动作来进行游戏。通过从原始屏幕像素到动作的策略学习，这些网络已在许多雅达利 2600 游戏中达到了最先进的性能。

然而，深度 Q 网络的局限在于，它们仅学习从有限数量的过去状态（在雅达利 2600 游戏中即游戏画面）中进行映射。实际训练中，DQN 使用的输入是由智能体最近遇到的四个状态组成的。因此，DQN 无法掌握那些需要玩家记住超过四个画面之前事件的游戏。换句话说，任何需要记忆超过四帧信息的游戏，在 DQN 看

[1] 原文：Matthew Hausknecht and Peter Stone. *Deep Recurrent Q-Learning for Partially Observable MDPs*. arXiv preprint arXiv:1507.06527, 2017. URL: https://arxiv.org/abs/1507.06527

[2] 译者：马燕琳。在原文基础上增补了部分推到细节，文中图片均为原文截屏。

来都将是非马尔可夫的，因为未来的游戏状态（及奖励）不仅仅依赖于 DQN 当前的输入。此时，游戏不再是马尔可夫决策过程（MDP），而是一个部分可观察的马尔可夫决策过程（POMDP）。

马尔可夫决策过程（MDP）

马尔可夫决策过程（MDP）是模拟智能体在**完全可观察**的**随机环境**中做序贯决策的数学模型，目标是找到最大化累积奖励的策略。其核心是**马尔可夫性质**，即下一状态仅取决于当前状态和动作，与历史无关。

MDP 通常用五元组 (S, A, P, R, γ) 表示：

- **状态集合** (S, State)：环境所有可能情况的集合。
- **动作集合** (A, Action)：智能体在每个状态可执行的操作的集合。
- **状态转移概率** (P, Transition Probability)：在状态 s 执行动作 a 后转移到状态 s' 的概率，记为 $P(s' \mid s, a)$。
- **奖励函数** (R, Reward Function)：在状态 s 执行动作 a 后转移到状态 s' 时获得的即时奖励，记为 $R(s, a, s')$ 或 $R(s, a)$。
- **折扣因子** (γ, Discount Factor)：取值范围为 $[0, 1]$，用于平衡即时奖励和未来奖励的重要性，γ 越接近 1 表示越重视长期回报。

MDP 的目标是找到一个**策略** (π, Policy)，即从状态到动作的映射，使得智能体执行该策略能获得最大的**期望累积折扣奖励**（回报）。求解 MDP 的经典算法包括**值迭代（Value Iteration）**、**策略迭代（Policy Iteration）**，以及**蒙特卡洛方法（Monte Carlo）**和**时序差分学习（Temporal-Difference Learning, TD）**（如 Q-Learning、SARSA）等无模型方法。

subsection* 部分可观测马尔可夫决策过程（POMDP）

部分可观察马尔可夫决策过程（POMDP）是 MDP 的扩展，用于建模智能体**无法直接观察到环境真实状态**的场景。智能体只能接收到一个与状态相关但具有不确定性的观测（Observation）。

POMDP 通常用六元组 (S, A, P, R, Ω, O) 表示：

- **状态集合** (S)、**动作集合** (A)、**转移概率** (P)、**奖励函数** (R)：与 MDP 中的含义相同。
- **观测集合** (Ω, Observation)：智能体所有可能感知到信息的集合。
- **观测函数** (O, Observation Function)：通常表示为 $O(o \mid s', a)$，表示在智能体执行动作 a 后环境转移到状态 s' 时，智能体接收到观测 o 的概率。

在 POMDP 中，智能体需要通过包含所有历史观察和动作的序列来维护一个

信念状态（Belief State），这是一个关于当前环境真实状态的概率分布。信念状态本身具有马尔可夫性，因此 POMDP 可以转化为一个以信念状态为状态的连续状态 MDP 来解决，但计算通常非常复杂。

表 1　MDP 与 POMDP 的核心区别

特征	MDP	POMDP
状态可见性	完全可观察（Fully Observable）	部分可观察（Partially Observable）
建模要素	状态、动作、转移概率、奖励、折扣因子	在 MDP 基础上增加观察集合和观察函数
智能体信息	知道环境的真实当前状态	不知道真实状态，仅有带噪声的局部观察
决策依据	当前状态 s_t	信念状态（基于历史动作和观察的概率分布）
问题复杂度	相对较低，有高效精确算法	非常高，通常需近似求解

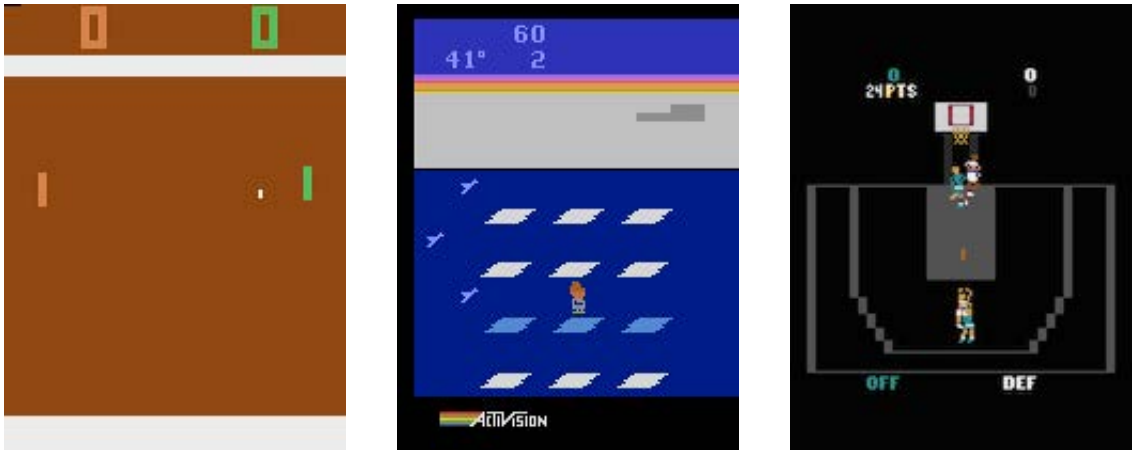
MDP 为在**完全信息**下进行序列决策提供了基础框架。而 POMDP 更贴近许多现实问题（如机器人导航、故障诊断），智能体只能获得**部分且有噪声的信息**，必须通过估计状态的概率分布（信念状态）来做出决策，因此建模和求解也更为复杂。

现实世界中的任务常常面临由于部分可观察性而导致的状态信息不完整和带有噪声的问题，正如 POMDP 中所体现的那样。如图 1 所示，如果只提供单帧游戏画面，许多雅达利 2600 游戏本质上就是 POMDP。一个例子是《Pong》游戏：当前画面只显示了球拍和球的位置，但无法显示球的速度。而知道球的行进方向是判断最佳球拍位置的关键因素。

我们观察到，当状态观测不完整时，DQN 的性能会下降，因此我们假设可以通过借鉴循环神经网络的研究进展，对 DQN 进行改进，使其更好地应对 POMDP 问题。为此，我们引入了深度循环 Q 网络（DRQN），它是长短期记忆网络（LSTM）（Hochreiter 和 Schmidhuber，1997）与深度 Q 网络的结合。关键的是，我们证明了 DRQN 能够处理部分可观测性问题，并且在评估阶段，当观测质量发生变化时，循环机制能够带来优势。

1.1 深度 Q 学习

强化学习（Sutton 和 Barto，1998）关注的是为与未知环境交互的智能体学习控制策略。这类环境通常被形式化为一个马尔可夫决策过程（MDP），它由一个四元组 (S, A, P, R) 完全描述。在每个时间步 t，与 MDP 交互的智能体观察一个状态 $s_t \in S$，并选择一个动作 $a_t \in A$，该动作决定了奖励 $r_t \sim R(s_t, a_t)$ 和下一个状态



(a) Pong (b) Frostbite (c) Double Drink

图 0.5　几乎所有的雅达利 2600（Atari 2600）游戏都包含运动的对象。如果只提供单帧输入图像，那么《Pong》（乒乓球）、《Frostbite》（冻伤）和《Double Dunk》（双重扣篮）都属于部分可观察马尔可夫决策过程（POMDP），因为单次观察无法揭示球（《Pong》和《Double Dunk》中）或浮冰（《Frostbite》中）的速度信息。

$s_{t+1} \sim P(s_t, a_t)$。

Q 学习（Watkins 和 Dayan，1992）用于估计在某个状态下执行某个动作的价值。这类价值估计被称为状态-动作值，或简称为 Q 值。Q 值通过迭代方式进行学习，将当前的 Q 值估计向观察到的奖励和结果状态 s' 的估计效用进行更新：

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1}$$

- $Q(s,a)$：**状态-动作值函数**（Q 值），表示在状态 s 下执行动作 a 后所能获得的预期累积折扣奖励。Q 值存储在 Q 表（Q-table）中，初始时通常设置为零或随机值。
- α（**学习率**）：取值范围为 $[0,1]$，控制新信息对旧 Q 值的影响程度。较大的 α 表示更注重新经验，较小的 α 则保留更多历史信息。
- r（**即时奖励**）：智能体在状态 s 执行动作 a 后环境反馈的瞬时奖励值。
- γ（**折扣因子**）：取值范围为 $[0,1]$，用于权衡当前奖励与未来奖励的重要性。γ 接近 1 表示智能体更重视长期回报，接近 0 则更关注即时收益。
- $\max_{a'} Q(s',a')$：表示在下一状态 s' 下，所有可能动作 a' 中对应的最大 Q 值。该项代表了对未来最优回报的估计，是 Q-learning 采用**离线策略**（off-policy）学习的体现（即智能体通过学习最优策略的行为来更新 Q 值，而实际行为可能基于探索性策略）)。
- s'：智能体执行动作 a 后环境转移到的下一个状态。

公式中的 $r + \gamma \max_{a'} Q(s',a')$ 称为 **TD 目标**（Temporal-Difference Target），而

$r + \gamma \max_{a'} Q(s', a') - Q(s, a)$ 称为 **TD 误差**（Temporal-Difference Error）。更新过程旨在使当前 Q 值 $Q(s, a)$ 向 TD 目标靠近，减少 TD 误差。

许多具有挑战性的领域（如雅达利游戏）包含过多独特状态，无法为每个 $S \times A$ 单独维护一个估计值。因此，通常使用一个模型来近似 Q 值（Mnih 等，2015）。在深度 Q 学习中，该模型是一个神经网络，其参数由权重和偏置组成，统一记为 θ。Q 值通过在前向传播后查询网络输出节点来在线估计，输入为状态，这种 Q 值记为 $Q(s, a \mid \theta)$。

与更新单个 Q 值不同，现在通过更新网络的参数来最小化一个可微的损失函数：

$$L(s, a \mid \theta_i) = \left(r + \gamma \max_{a'} Q(s', a' \mid \theta_i) - Q(s, a \mid \theta_i)\right)^2 \tag{2}$$

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta L(\theta_i) \tag{3}$$

由于 $|\theta| \ll |S \times A|$，神经网络模型能够自然地泛化到未训练过的状态和动作上。然而，由于同一个网络既用于生成下一状态的目标 Q 值，又用于更新当前 Q 值，这种更新可能会对其他 Q 值估计产生负面影响，导致性能震荡甚至发散（Tsitsiklis 和 Roy，1997）。

深度 Q 学习采用三种技术来恢复学习的稳定性：

1. 首先，将经验 $e_t = (s_t, a_t, r_t, s_{t+1})$ 存储在一个经验回放记忆库 $\mathcal{D}$ 中，并在训练时从中均匀采样。
2. 其次，使用一个独立的目标网络 $\hat{Q}$ 来为主网络提供“滞后”的更新目标。目标网络的作用是部分解耦网络自身生成目标所带来的反馈。$\hat{Q}$ 与主网络结构相同，区别在于其参数 θ^- 每 10,000 次迭代才更新一次以匹配主网络参数 θ。
3. 最后，使用自适应学习率方法（如 RMSProp（Tieleman 和 Hinton，2012）或 ADADELTA（Zeiler，2012））为每个参数维护一个独立的学习率 α，并根据该参数的梯度更新历史进行调整。

具体而言，在每次训练迭代 i 时，从经验回放库 $\mathcal{D}$ 中均匀采样一条经验 $e_t = (s_t, a_t, r_t, s_{t+1})$。网络的损失函数定义如下：

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(y_i - Q(s, a; \theta_i)\right)^2 \right] \tag{4}$$

其中 $y_i = r + \gamma \max_{a'} \hat{Q}(s', a'; \theta^-)$ 是由目标网络 $\hat{Q}$ 提供的滞后更新目标。经验表明，以这种方式进行的更新是可行且稳定的（Mnih 等，2015）。

1.2 部分可观察性

在现实世界环境中，智能体很少能够获得系统的完整状态，甚至无法确定该状态。换句话说，马尔可夫性质在现实世界环境中几乎不成立。部分可观察马尔可夫决策过程（POMDP）通过明确指出智能体所接收到的感知只是底层系统状态的部分片段，更好地捕捉了许多现实世界环境的动态特性。

形式上，一个 POMDP 可以用一个六元组 (S, A, P, R, Ω, O) 来描述。其中 S、A、P、R 分别表示状态、动作、转移概率和奖励函数，正如之前所述，区别在于智能体不再能够得知真实的系统状态，而是接收一个观察值 $o \in \Omega$。该观察值是根据底层系统状态生成的，遵循概率分布 $o \sim O(s)$。

原始版本的深度 Q 学习并没有显式机制来推断 POMDP 的底层状态，它仅在观察值能够反映底层系统状态时才有效。在一般情况下，从观察值中估计 Q 值可能会非常不准确，因为 $Q(o, a \mid \theta) \neq Q(s, a \mid \theta)$。

我们的实验表明，在深度 Q 学习中引入循环机制可以使 Q 网络更好地估计底层系统状态，从而缩小 $Q(o, a \mid \theta)$ 与 $Q(s, a \mid \theta)$ 之间的差距。换句话说，循环深度 Q 网络能够更好地从一系列观察值中逼近真实的 Q 值，从而在部分可观察的环境中学习到更优的策略。

2 DRQN 架构

如图 2 所示，DRQN 的架构将 DQN 的第一个全连接层替换为一个长短期记忆网络（LSTM）（Hochreiter 和 Schmidhuber，1997）。在输入方面，该循环网络仅使用一张 84×84 的预处理图像，而不是 DQN 所需的前四帧图像。

第一个隐藏层使用 32 个 8×8 的卷积核，以步长 4 在输入图像上进行卷积，并应用修正线性激活函数（rectifier）。第二个隐藏层使用 64 个 4×4 的卷积核，以步长 2 进行卷积，同样后面是一个修正线性激活函数。第三个隐藏层使用 64 个 3×3 的卷积核，以步长 1 进行卷积，之后也接一个修正线性激活函数。

卷积层的输出被送入一个全连接的长短期记忆（LSTM）层。最后，一个全连接的线性层输出每个可能动作对应的 Q 值。我们在尝试了多种结构变体之后，最终确定了这一架构；详见附录 A。

3 稳定的循环更新机制

对一个包含循环结构和卷积层的网络进行更新，需要在每次反向传播中包含多个时间步的游戏画面和目标值。此外，LSTM 的初始隐藏状态可以被清零，也可

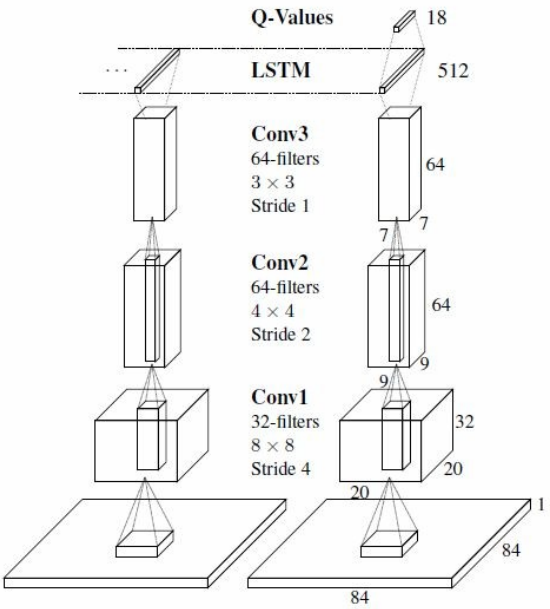


图 0.6　DRQN 对单通道游戏画面图像进行三次卷积操作。得到的激活值通过 LSTM 层在时间维度上进行处理。此处展示了最后两个时间步的情况。LSTM 的输出在通过全连接层后转化为 Q 值。卷积滤波器以带尖顶的矩形子框形式描绘。

以从上一个状态中继承。我们考虑了两种更新方式：

1. **引导式顺序更新**：从经验回放库中随机采样完整的回合（episode），从回合的起点开始更新，并按时间顺序推进至回合结束。每个时间步的目标值由目标 Q 网络 $\hat{Q}$ 生成。RNN 的隐藏状态在整个回合中持续传递。**这种方式试图最大化利用 LSTM 的记忆能力，但训练数据相关性强，违背了随机梯度下降（SGD）及其变体（如 Adam）的有效性，依赖的基本假设：训练数据是独立同分布的（IID, Independent and Identically Distributed）**
2. **引导式随机更新**：从经验回放库中随机采样回合，并从回合中的任意位置开始更新，仅展开固定数量的时间步（例如一次反向传播）。每个时间步的目标值由目标 Q 网络 $\hat{Q}$ 生成。RNN 的初始状态在每次更新开始时会被清零。**这种方式试图在保持 LSTM 功能的同时，尽可能符合随机采样要求。**

顺序更新的优点在于可以从回合开始时就持续传递 LSTM 的隐藏状态。然而，通过按顺序采样整个回合的经验，这种方式违背了 DQN 所采用的随机采样策略。

随机更新更符合经验随机采样的原则，但代价是每次更新时必须将 LSTM 的隐藏状态清零。这种清零操作会使得 LSTM 难以学习那些时间跨度超过反向传播时间步数的函数（无法获取和利用比这段序列更长的时间历史信息）。

实验表明，这两种更新方式都是可行的，并在多个游戏中收敛到性能相近的策略。因此，为了简化实现，本文中的所有结果均采用了随机更新策略。我们预

期，所有呈现的结果也适用于顺序更新策略。

4 雅达利游戏：是 MDP 还是 POMDP？

雅达利 2600 游戏机的状态完全可以由其 128 字节的内存描述和表征。然而，人类玩家和智能体只能观察到游戏机生成的画面。对许多游戏而言，仅凭单帧画面并不足以判断系统的完整状态。DQN 通过将状态表示扩展为最近四帧游戏画面，从而推断出雅达利游戏的完整状态。许多原本属于 POMDP 的游戏，在这种设定下就变成了 MDP。在 (Mnih 等，2015) 研究的 49 款游戏中，作者未能发现任何一款在四帧输入下仍然是部分可观察的[3]。为了在不减少 DQN 输入帧数的前提下，向雅达利游戏中引入部分可观察性，我们对经典游戏《Pong》进行了一种特定的修改。

5 闪烁版 Pong POMDP

我们引入了“闪烁版 Pong POMDP”——这是对经典《Pong》游戏的修改，使得在每个时间步，画面以 0.5 的概率被完全显示或被完全遮蔽。通过这种方式随机遮蔽画面，使得玩家无法完整观察游戏过程，从而将 Pong 转变为一个 POMDP。

要在闪烁版 Pong 中取得成功，必须跨帧整合信息，以估计关键变量，例如球的位置与速度，以及球拍的位置。由于预期有一半的画面会被遮蔽，成功的玩家必须能够应对连续多帧画面被遮蔽的情况。

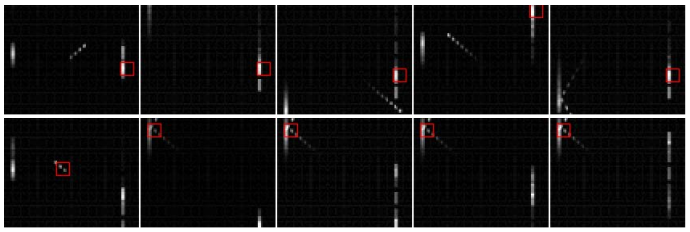
我们训练了三种网络来玩闪烁版 Pong：一种是循环结构的单帧 DRQN，一种是标准的四帧 DQN，还有一种是扩展的十帧 DQN。如图 4a 所示，为 DQN 提供更多帧数的历史信息可以提升其表现。然而，即使是使用了十帧历史的 DQN，仍然难以获得正分数。

使用历史游戏画面最重要的作用之一，是可以通过卷积操作检测物体的速度。图 3 展示了激活不同卷积滤波器最强的游戏画面，证实了十帧 DQN 的滤波器确实可以检测物体的速度，尽管其可靠性可能不如在正常无遮蔽版 Pong 中那么高[4]。

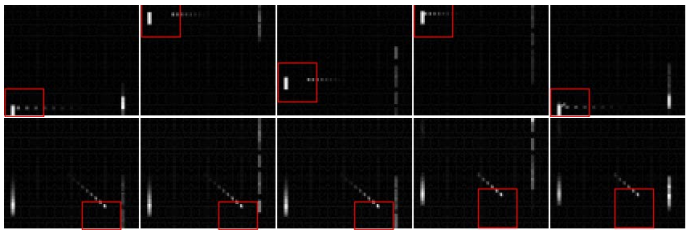
值得注意的是，即使每个时间步仅输入单帧图像，DRQN 在此任务中依然表现良好。在单帧输入下，DRQN 的卷积层无法检测任何速度信息。因此，必须依靠更高层的循环结构来弥补闪烁画面带来的信息缺失以及卷积层无法检测速度的

[3]有些雅达利游戏无疑属于部分可观测马尔可夫决策过程（POMDP），例如 Blackjack（二十一点），因为在该游戏中，庄家的牌是隐藏的，玩家无法看到。但遗憾的是，ALE（Arcade Learning Environment）模拟器并不支持 Blackjack 这款游戏。

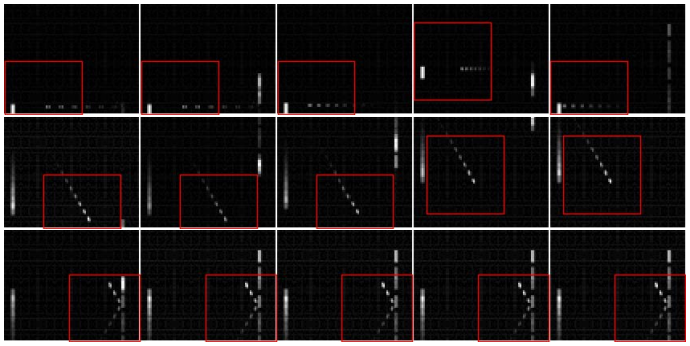
[4](Guo et al. 2014) 的研究也证实了，卷积滤波器能够学会响应游戏对象中观察到的运动模式。



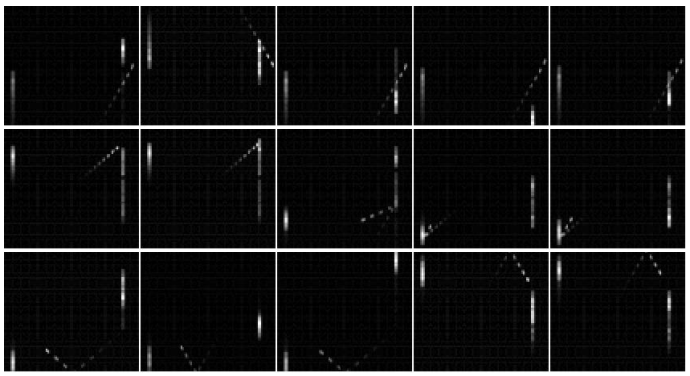
(a) 第一卷积层的滤波器



(b) 第二卷积层的滤波器



(c) 第三卷积层的滤波器



(d) 最大化三个示例 LSTM 单元激活的图像序列

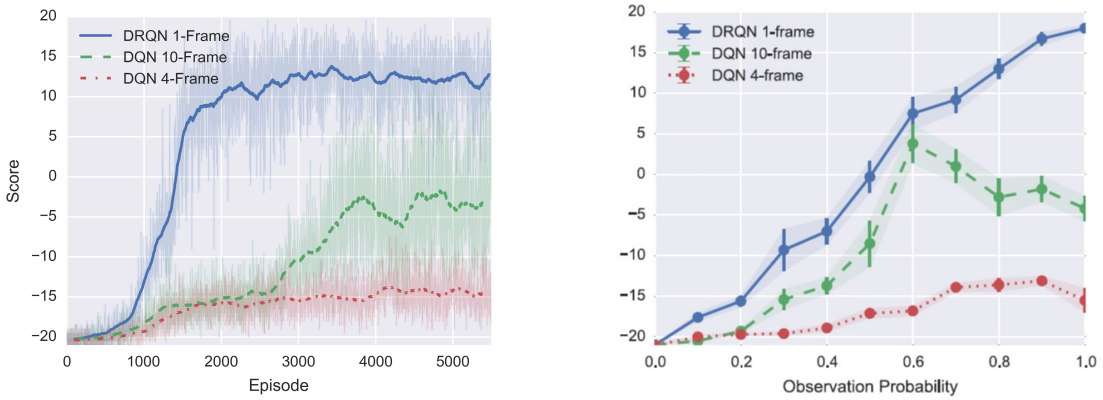
图 0.7　在《Pong》游戏中，由 10 帧输入训练的 DQN 所学习到的卷积滤波器示例。每一行展示了能够触发指定层中特定卷积滤波器达到最大激活的输入帧。红色边界框标出了导致最大激活的输入图像区域。第一卷积层中的大多数滤波器仅检测球拍。第二卷积层（Conv2）的滤波器开始检测特定方向的球体运动，部分滤波器同时跟踪球和球拍。几乎所有的第三卷积层（Conv3）滤波器都跟踪球与球拍的交互，包括球的偏转、速度及运动方向。尽管每次只能看到单帧图像，但各个 LSTM 单元仍能分别检测高级事件：智能体未能接到球、球从球拍上反弹以及球从墙壁反弹。每张图像叠加了智能体最近看到的 10 帧画面，近期帧的亮度更高。

问题。尽管如此，DRQN 仍然能在满分为 21 分的游戏中经常获得超过 10 分的成绩。图 3d 进一步证实，LSTM 层中的单个单元能够通过时间整合带有噪声的单帧信息，从而识别出高级别的 Pong 游戏事件，例如玩家未接到球、球碰到球拍反弹，或球撞到墙壁反弹。

DRQN 通过时间反向传播（BPTT）训练，展开最近十个时间步。因此，非循环的十帧 DQN 与循环的单帧 DRQN 都能接触到相同的游戏画面历史[5]。DRQN 能更有效地利用这段有限的历史信息，从而获得更高的分数。

因此，在处理部分可观测性问题时，存在一个选择：是使用一个带有长历史观测信息的非循环深度网络，还是使用一个在每个时间步仅接收单次观测进行训练的循环网络。闪烁 Pong（Flickering Pong）提供了一个例证：即使循环深度网络所能获取的过去观测信息数量与非循环网络相同，其表现也更优。DRQN 和 DQN 的性能在一组十款游戏中进行了进一步比较（表 3），结果显示，两种算法并未表现出系统性的优劣性区别。

6 泛化性能



(a) 闪烁 Pong　　(b) 策略泛化

图 0.8　左图：在部分可观的闪烁 Pong 环境中，尽管每次只能看到单帧输入，深度循环 Q 网络（DRQN）处理感官噪声的能力远胜于深度 Q 网络（DQN）。由于缺乏循环机制，依赖 4 帧历史画面的 DQN 难以克服由闪烁游戏屏幕引起的部分可观察性问题。不过，为 DQN 提供 10 帧历史信息后，其性能得到部分改善。右图：在经过观察概率为 0.5（即每秒画面有 50% 的概率被遮挡）的训练后，测试所学策略的泛化能力。DRQN 学习到的策略能够很好地泛化到不同的观察概率（即不同程度的遮挡）下。相比之下，DQN 的性能在略高于其训练时所用概率（0.5）的观察概率处达到峰值，随后下降，泛化能力较弱。图中的误差线表示标准误（Standard Error）。

为了分析这些闪烁版 Pong 智能体的泛化性能，我们在改变画面遮蔽概率的条

[5]然而，(Karpathy, Johnson, and Li 2015) 的研究表明，LSTM（长短期记忆网络）能够在训练时学习基于有限时间步的函数，并在测试时将其泛化到更长的序列上。

件下，评估了 DRQN、十帧 DQN 和四帧 DQN 的最佳策略。需要注意的是，这些策略都是在闪烁概率为 p = 0.5 的条件下训练得到的，而现在我们将在不同的 p 值下进行测试。图 4b 显示，随着观察概率的增加，DRQN 的性能持续提升。相比之下，十帧 DQN 的性能在接近其训练时的观察概率附近达到峰值，随后即使观察概率进一步提高，其性能反而下降。因此，DRQN 学习到的策略能够使其性能随观察质量的提升而改善。这一特性对于那些观察质量会随时间变化的领域而言，具有重要的价值。

7 DRQN 能否模拟无遮蔽情况下的玩家行为？

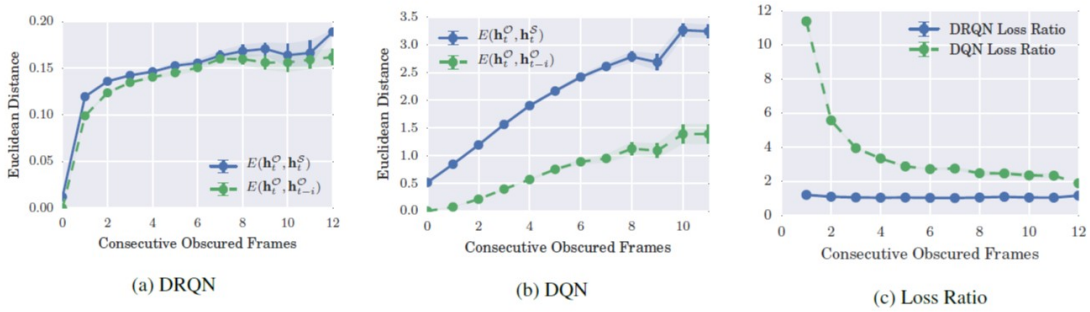


图 0.9　左侧两个子图展示了无遮蔽网络与被遮蔽网络的激活值之间的平均欧几里得距离 $E(h_O^t, h_S^t)$ 随连续被遮蔽游戏画面数量变化的函数关系，横坐标为 人为地、连续地遮挡住游戏屏幕（或输入信息）的帧数，旨在评估当智能体连续多帧完全无法获取当前环境信息时，它的表现会多差。其中，$E(h_O^t, h_O^{t-i})$ 表示网络当前时间步的激活值与上一次观察到屏幕画面时的激活值之间的欧几里得距离。右侧子图展示了这两个量的比值，结果表明当游戏画面被遮蔽时，DRQN 的激活值变化模式比 DQN 的激活值更接近于无遮蔽玩家的激活值变化模式。

为了更好地理解 DRQN 能在闪烁版 Pong 环境中表现良好的机制，我们分析了在相同游戏轨迹下，分别在 $p = 0.5$（闪烁）和 $p = 1.0$（无遮蔽）条件下，DRQN 玩家内部激活值的差异。我们假设：如果该智能体能在画面被遮蔽时推断出底层游戏状态，那么它的激活值应与在无遮蔽条件下运行的智能体相似。

我们使用归一化的欧几里得距离作为度量标准，来衡量两个激活向量 u 和 v 之间的差异：

$$E(u, v) = \frac{2}{\|u\|_1} \|u - v\|_2^2 \tag{5}$$

我们记 DRQN 的 LSTM 层在时间步 t 的激活值为 h_t^o（在闪烁条件下）。当关闭闪烁并显示完整画面时，记其激活值为 h_t^s。

我们让训练好的闪烁版 Pong 玩家进行 10 局游戏，并记录其 LSTM 层激活值序列 $[h_1^o, \ldots, h_t^o]$。接着，在相同的游戏轨迹下，我们将所有画面设为可见，再次运行该网络，得到激活值序列 $[h_1^s, \ldots, h_t^s]$。如果 LSTM 能够在画面未被观察到时推

断出游戏状态，那么对于所有画面被遮蔽的时间步 t，$E(h_t^o, h_t^s)$ 的平均欧几里得距离应当较低。图 5a–5b 展示了该距离随连续遮蔽帧数增加的变化趋势，分别对比了 DRQN 和十帧 DQN[6]。与 DQN 相比，DRQN 在闪烁版 Pong 和无遮蔽 Pong 之间的激活值平均距离更小。此外，即便在存在噪声的情况下，DRQN 的激活值变化也更加平滑。

然而，仅仅比较欧几里得距离并不能说明全部问题，因为 10 帧 DQN 的激活值在帧与帧之间的预期变化量可能本身就比 DRQN 更大。为了控制这一变量，图 5a-5b 额外绘制了自上一帧被观测到以来激活值的总体变化，即 $E(h_O^t, h_O^{t-i})$。该度量提供了激活值在时间步与时间步之间整体波动性的一个直观感受。

最终，当屏幕被遮蔽时，网络的激活值预期会发生变化。我们关注的是它们是否沿着与未遮蔽网络激活值相同的方向变化。令 A_i 表示轨迹中所有最近连续 i 帧屏幕被遮蔽的时间步集合。我们随后计算了一个平均比率：该比率的分子是遮蔽网络与未遮蔽网络激活值之间的距离，分母是当前时间步的遮蔽激活值与游戏屏幕最后一次被观测到的时间步的激活值之间的距离。

$$\text{LossRatio}_i = \frac{1}{|A_i|} \sum_{t \in A_i} \frac{E(h_t^s, h_t^o)}{E(h_t^o, h_t^{-i})} \tag{6}$$

该值越低，说明被遮蔽网络的激活值变化方向越接近无遮蔽网络。如图 5c 所示，在所有连续遮蔽帧数下，DRQN 的 loss ratio 均低于 DQN，表明它更能模仿无遮蔽情况下的网络激活行为。事实上，DRQN 在画面被遮蔽时的激活值与无遮蔽激活值以及最后一次可见画面的激活值大致等距，这意味着 DRQN 在遭遇遮蔽画面时，无法完全推断出隐藏的游戏状态，但也不会完全保留过去的激活值。

8 在标准雅达利游戏上的评估

我们选取了以下九款雅达利游戏进行评估：

在提供最近四帧输入的条件下，这些游戏都属于 MDP 而非 POMDP。因此，我们没有理由预期 DRQN 会优于 DQN。确实，表 2 中的结果表明，平均来看，DRQN 的表现与 DQN 大致相当。具体而言，我们重新实现的 DQN 与原始版本表现相近，在九款游戏中有五款超过了原始成绩，但在《Centipede》和《Chopper Command》上得分不足原始版本的一半。DRQN 在《Frostbite》和《Double Dunk》两款游戏上表现优于我们实现的 DQN，但在《Beam Rider》上明显更差（见图 6）。《Frostbite》（见图 1b）要求玩家跳过四排移动的冰山，并返回屏幕顶部。在多次穿越冰山后，玩

[6]由于 10 帧 DQN 没有 LSTM 层，因此其欧几里得距离是在第一个卷积后的全连接层的激活值上计算的。该层具有 512 个单元，与 DRQN 的 LSTM 层单元数相同。

	DRQN ±std		DQN ±std
Game		**Ours**	Mnih et al.
小行星 Asteroids	1020 (±312)	1070 (±345)	1629 (±542)
光束骑士 Beam Rider	3269 (±1167)	**6923** (±1027)	6846 (±1619)
保龄球 Bowling	62 (±5.9)	72 (±11)	42 (±88)
蜈蚣 Centipede	3534 (±1601)	3653 (±1903)	8309 (±5237)
直升机控制台 Chopper Cmd	2070 (±875)	1460 (±976)	6687 (±2916)
双重扣篮 Double Dunk	**-2** (±7.8)	−10 (±3.5)	−18.1 (±2.6)
冻伤 Frostbite	**2875** (±535)	519 (±363)	328 (±250.5)
冰球 Ice Hockey	−4.4 (±1.6)	−3.5 (±3.5)	−1.6 (±2.5)
吃豆人小姐 Ms. Pacman	2048 (±653)	2363 (±735)	2311 (±525)

表 2　在标准雅达利游戏上，DRQN 的性能与 DQN 相当，在《Frostbite》和《Double Dunk》游戏中表现优异，但在《Beam Rider》游戏中表现不佳。粗体字表示 DRQN 与我们的 DQN 之间存在统计显著性差异。

家收集到足够的冰块，可以在右上角建造一个冰屋，随后进入冰屋以进入下一关。如图 6 所示，在大约 12,000 局训练后，DRQN 学会了一种策略，能够稳定地通过《Frostbite》的第一关。实验细节见附录 C。

9 从 MDP 到 POMDP 的泛化

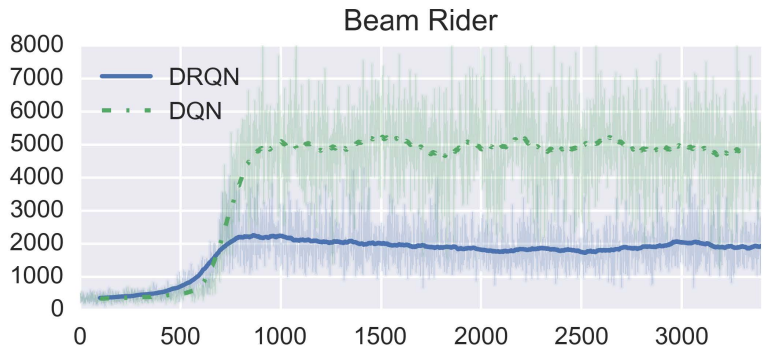
得研究的问题是反过来的情况：一个循环网络能否在标准 MDP 上训练，并在评估时泛化到 POMDP？为回答这个问题，我们在表 1 中所有 9 款游戏的闪烁版本上，评估了 DRQN 和 DQN 的最高得分策略。图 7 显示，尽管两种算法都因信息缺失而显著性能下降，但在所有闪烁程度下，DRQN 相比 DQN 保留了更多原有性能。我们由此得出结论：循环控制器对信息缺失具有一定的鲁棒性，即使它们是在完整状态信息下训练的。

10 相关研究

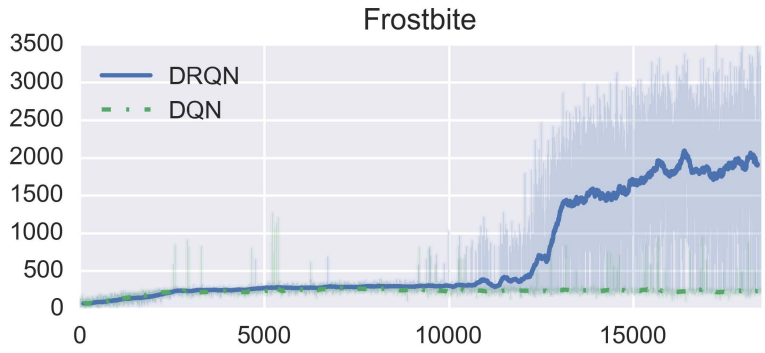
此前，已有研究表明，在使用策略梯度方法训练时，LSTM 网络能够求解 POMDP（Wierstra 等，2007）。与策略梯度不同，本文使用时序差分更新来引导动作价值函数的学习。此外，通过联合训练卷积层与 LSTM 层，我们能够直接从像素中学习，而无需手工设计特征。

LSTM 也曾被用作优势函数近似器，并显示出在部分可观察的走廊任务和倒立摆任务中，其表现优于不含 LSTM 的循环神经网络（Bakker，2001）。尽管原理相似，但这些任务的状态空间极小，仅包含少量特征。

与我们的工作并行，Narasimhan、Kulkarni 与 Barzilay（2015）也独立地将 LSTM 与深度强化学习结合，证明循环结构有助于更好地游玩基于文本的奇幻游戏。其



(a)



(b)

图 0.10　《冻伤》（Frostbite）和《光束骑士》（Beam Rider）分别代表了 DRQN 表现最佳和最差的游戏。当智能体学会可靠地通过第一关时，《冻伤》的游戏性能出现显著提升。

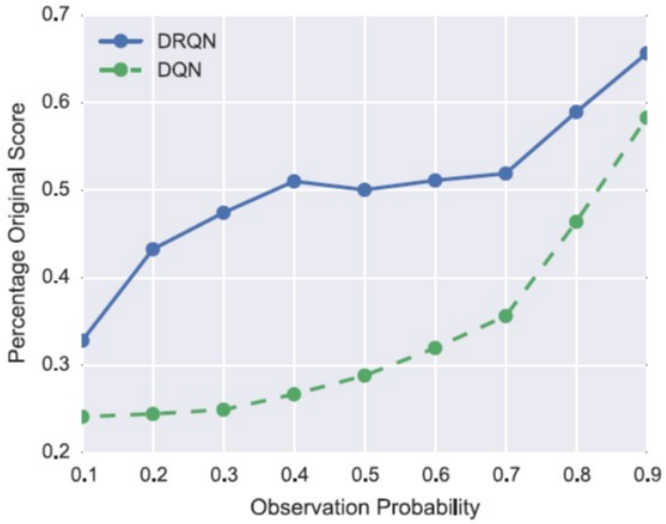


图 0.11　当在正常游戏（MDPs）上训练，然后在闪烁游戏（POMDPs）上评估时，DRQN 的性能下降比 DQN 更为平缓。图中每个数据点显示了表 1 中所有 9 款游戏得分的平均百分比（相对于原始游戏得分）。

方法与本研究类似，但领域不同：尽管自动生成的奇幻文本表面上复杂，其底层 MDP 的状态空间流形维度较低，两款游戏中较复杂的一款仅包含 56 个底层状态。相比之下，雅达利游戏的状态空间丰富得多，典型游戏拥有数百万种不同状态；然而文本游戏的动作空间远大于雅达利，分支因子达到 222，而雅达利仅为 18。

Flickering	**DRQN ±std**	**DQN ±std**
小行星 Asteroids	1032 (±410)	1010 (±535)
光束骑士 Beam Rider	618 (±115)	**1685.6** (±875)
保龄球 Bowling	65.5 (±13)	57.3 (±8)
蜈蚣 Centipede	4319.2 (±4378)	5268.1 (±2052)
直升机控制台 Chopper Cmd	1330 (±294)	1450 (±787.8)
双重扣篮 Double Dunk	-14 (±2.5)	-16.2 (±2.6)
冻伤 Frostbite	414 (±494)	436 (±462.5)
冰球 Ice Hockey	-5.4 (±2.7)	-4.2 (±1.5)
吃豆人小姐 Ms. Pacman	1739 (±942)	1824 (±490)
乒乓球 Pong	**12.1** (±2.2)	**-9.9** (±3.3)

表 3　每个屏幕以 0.5 的概率被遮挡，形成了标准游戏的部分可观测闪烁版本。粗体表示统计显著性的试验部分。

[7]

11 讨论与结论

现实世界任务常常因部分可观察性而导致状态信息不完整且带有噪声。我们通过将长短期记忆（LSTM）与深度 Q 网络相结合，对 DQN 进行修改，使其能够处理 POMDP 中典型的带噪观察。由此产生的深度循环 Q 网络（DRQN）尽管在每个时间步只能看到单帧画面，仍能通过跨帧整合信息，检测出屏幕上物体的速度等关键信息。此外，在 Pong 游戏中，DRQN 比标准深度 Q 网络更擅长应对由画面闪烁引起的部分可观察性。

进一步分析表明，当在部分观察条件下训练时，DRQN 学到的策略可以泛化到完全观察的情形。在闪烁版 Pong 任务中，其性能随可观察性提高而提升，当所有画面均可见时几乎达到完美水平。这表明循环网络学到的策略既足够鲁棒以应对缺失画面，又具备可扩展性，能够在数据增多时持续提升性能。

泛化能力也体现在相反方向：在标准雅达利游戏上训练后，再在闪烁版游戏上测试时，DRQN 在所有部分信息水平下的性能均优于 DQN。

实验结果表明，Pong 在我们考察的游戏中属于特例。在十款闪烁版 MDP 任务中，我们并未观察到使用循环结构带来系统性提升。同样，在非闪烁的雅达利

[7]使用独立样本 t 检验和本杰米尼-霍奇伯格（Benjamini-Hochberg）程序，在显著性水平 P = 0.05 下确定得分的统计显著性。

游戏中，循环玩家与非循环玩家之间也几乎没有显著差异。这一观察使我们得出结论：尽管循环结构是一种处理多帧观察的可行方法，但与将观察帧堆叠在卷积网络输入层相比，它并未带来系统性优势。

未来的一个有趣研究方向是找出 Pong 和 Frostbite 等任务中使循环结构表现更优的关键特征。

12 致谢

本研究在德克萨斯大学奥斯汀分校人工智能实验室的“学习智能体研究组”（LARG）完成。LARG 的研究部分得到以下资助：

- 美国国家科学基金会（CNS-1330072、CNS-1305287）
- 美国海军研究办公室（ONR 21C184-01）
- 美国空军研究实验室（AFRL FA8750-14-1-0070）
- 美国空军科学研究办公室（AFOSR FA9550-14-1-0087）
- Yujin Robot 公司

此外，德克萨斯高级计算中心（TACC）与 Nvidia 亦提供了额外支持。

附录 A: 替代架构

描述	性能提升百分比
LSTM 替换 IP1	709%
ReLU-LSTM 替换 IP1	533%
LSTM 置于 IP1 之上	418%
ReLU-LSTM 置于 IP1 之上	0%

在游戏《Beam Rider》上评估了多种替代架构。我们探索了两种可能性：要么将第一个非卷积全连接层替换为 LSTM 层（LSTM 替换 IP1），要么在第一个和第二个全连接层之间添加 LSTM 层（LSTM 置于 IP1 之上）。结果强烈表明 LSTM 应该替换 IP1。我们假设这允许 LSTM 直接访问卷积特征。此外，在 LSTM 层之后添加 ReLU 层会持续降低性能。

附录 B: 计算效率

RNN 的计算效率是一个重要问题。我们进行了实验，执行 1000 次前向和反向传播，并报告每次传播所需的平均时间（毫秒）。实验使用单个 Nvidia GTX Titan Black，配合 CuDNN 和完全优化的 Caffe 版本。结果表明，计算量在输入层堆叠帧

数和展开迭代次数两方面都呈亚线性增长。即便如此，在大量堆叠帧数上训练并展开多次迭代的模型通常在计算上是不可行的。例如，一个展开 30 次迭代且使用 10 个堆叠帧的模型将需要超过 56 天才能达到 1000 万次迭代。

	反向传播 (ms)			前向传播 (ms)		
	帧数			帧数		
	1	4	10	1	4	10
基线	8.82	13.6	26.7	2.0	4.0	9.0
展开 1	18.2	22.3	33.7	2.4	4.4	9.4
展开 10	77.3	111.3	180.5	2.5	4.4	8.3
展开 30	204.5	263.4	491.1	2.5	3.8	9.4

表 3: 每次前向/反向传播的平均毫秒数。“帧数”指输入图像的通道数。“基线”是非循环网络（例如 DQN）。“展开”指通过时间反向传播 1/10/30 步的 LSTM 网络。

附录 C：实验细节

策略每 50,000 次迭代评估一次，评估方式为运行 10 个回合并取得分平均值。网络训练持续进行 1000 万次迭代，并使用大小为 400,000 的回放内存。此外，所有网络均使用 ADADELTA（Zeiler, 2012）优化器，学习率设置为 0.1，动量（momentum）为 0.95。LSTM 的梯度被裁剪（clipped）至 10 以确保学习稳定性。所有其他设置均与（Mnih 等, 2015）文中所述保持一致。

所有网络均使用 Arcade 学习环境（ALE）（Bellemare 等, 2013）进行训练。所使用的 ALE 选项包括：颜色平均（color averaging）、最小动作集（minimal action set）以及死亡检测（death detection）。

DRQN 在 Caffe（Jia 等, 2014）中实现。源代码可在以下网址获取：`https://github.com/mhauskn/dqn/tree/recurrent`。

参考文献

[1] **Reinforcement learning with long short-term memory**. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1475–1482. MIT Press, 2001.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. *The arcade learning environment: An evaluation platform for general agents*. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[3] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3338–3346. Curran Associates, Inc., 2014.

[4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[6] A. Karpathy, J. Johnson, and F.-F. Li. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[8] K. Narasimhan, T. Kulkarni, and R. Barzilay. Language understanding for text-based games using deep reinforcement learning. *CoRR abs/1506.08941*, 2015.

[9] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[10] T. Tieleman and G. Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

[11] 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.

[12] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[13] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *Proceedings of the 2007 International Conference on Artificial Neural Networks (ICANN)*, 2007.

[14] M. D. Zeiler. ADADELTA: An adaptive learning rate method. *CoRR abs/1212.5701*, 2012.

深度强化学习中的对决网络架构[1,2]

摘要

近年来，深层表征在强化学习中的应用取得了许多成功。然而，这些应用中很多仍然依赖传统架构，例如卷积网络、长短期记忆网络 (LSTM) 或自编码器。本文提出了一种新的无模型强化学习神经网络架构。我们的对决网络 (dueling network) 表示两个独立的估计器：一个估计状态价值函数，另一个估计状态相关的动作优势函数。这种分解的主要好处在于，在不改变底层强化学习算法的前提下，可以实现跨动作的泛化学习。我们的实验结果表明，存在多个具有相似价值的动作时，该架构能够更好地进行策略评估。此外，对决架构使得我们的 RL 智能体在雅达利 (Atari) 2600 游戏中超越了最优水平。

1 引言

近年来，深度学习极大推动了机器学习在可扩展性与性能方面的进步[1]。强化学习 (RL)) 和控制中的序列决策问题是其中一个令人兴奋的应用方向。代表性成果包括深度 Q 学习[2]、深度视觉运动策略[3]、带注意力机制的循环网络[4]、以及带有嵌套的模型预测控制[5]。其他成功结果还包括大规模并行框架[6] 和围棋专家预测系统[7]，这些方法能够生成与蒙特卡洛树搜索程序相媲美的策略，更在结合搜索算法后，成功击败了职业棋手[8]。

尽管如此，当前大多数强化学习方法仍采用如卷积神经网络、多层感知机 (MLPs)、长短期记忆网络 (LSTM) 和自动编码器等标准神经网络架构。近年来的研究重点主要集中在设计改进型控制算法与强化学习算法，或是简单地将现有神经网络架构整合到强化学习方法中。本文采取一种创新且互补的方法，专注于开发更适合无模型强化学习的神经网络架构。这种研究思路的优势在于：新架构能够轻松地与现有和未来的强化学习算法相结合。具体而言，本文提出了一种新的网络架构 (图 1)，但仍基于已公开的强化学习算法实现。

我们提出的网络架构，称为对决架构，明确地分离了状态值和 (依赖于状态的) 动作优势值的表示。对决架构由两个分别代表价值和优势函数的流组成，同时共

[1]原文：Ziyu W, Tom S, Matteo H, et al. Dueling Network Architectures for Deep Reinforcement Learning [J]. arXiv preprint, arXiv:1511.06581v3, 2016.

[2]译者：葛涵。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

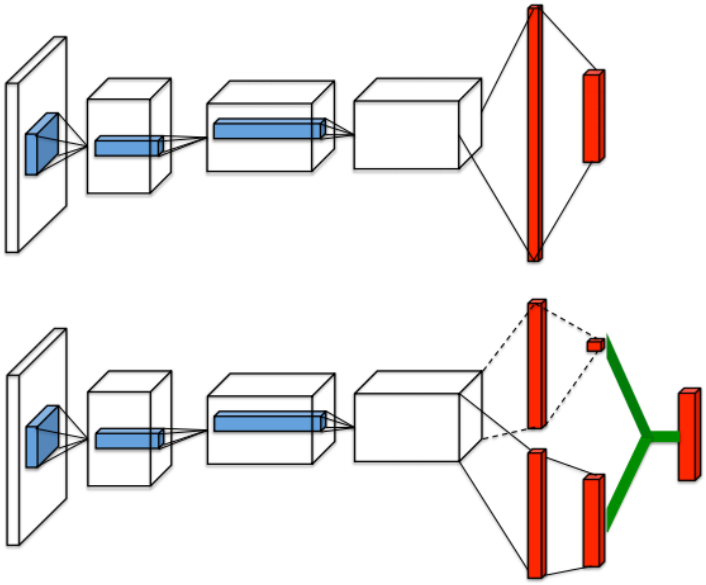


图 0.12　一个流行的单流 Q 网络 (上方) 和对决 Q 网络 (下方)。对决网络有两个流，分别估计 (标量) 状态值和每个动作的优势值；绿色输出模块通过方程 (9) 将它们组合。两个网络均为每个动作输出 Q 值。

享一个公共的卷积特征学习模块。如图 1 所示，两个信息流通过一个特殊的聚合层进行组合，生成状态-动作价值函数 Q 的估计值。这种对决网络架构应理解为：用带有两个信息流的单一 Q 网络替代现有算法，如深度 Q 网络 (DQN)[2] 中流行的单流 Q 网络。该架构无需额外监督，即可分别自动生成状态价值函数和优势函数的独立估计值。

直观来说，对决架构能够学习哪些状态具有价值 (或不具备价值)，而无需为每个状态单独学习动作效果。这种特性在动作对环境无显著影响的状态下尤为实用。以图 2 所示的显著性图为例[1]：这些图像是根据 Simonyan 等人[9] 提出的方法，通过计算训练好的价值流和优势流在输入视频中的雅可比矩阵而生成的。(实验部分对此方法有更详细说明) 该图展示了两个不同时间步的价值流和优势流的显著性图。在第一个时间步 (最左侧图像对) 中，我们可以看到价值网络流关注道路环境，尤其是出现新车的地平线区域，并且关注得分信息。而另一侧的优势流则对视觉输入关注度较低，因为当前方没有车辆时，其动作选择实际上无关紧要。然而，在第二个时间步 (最右边的一对图像) 中，优势流关注一辆紧挨的汽车，这使得它的动作选择变得相关。

在实验中，我们证明了当冗余或相似的动作被添加到学习问题中时，对决架构可以在策略评估期间更快地识别正确的动作。

我们还在极具挑战性的雅达利 2600 测试平台上，评估了这种对决架构带来的

[1]https://www.youtube.com/playlist?list=PLVFXyCSfS2Pau0gBh0mwTxDmutywWyFBP

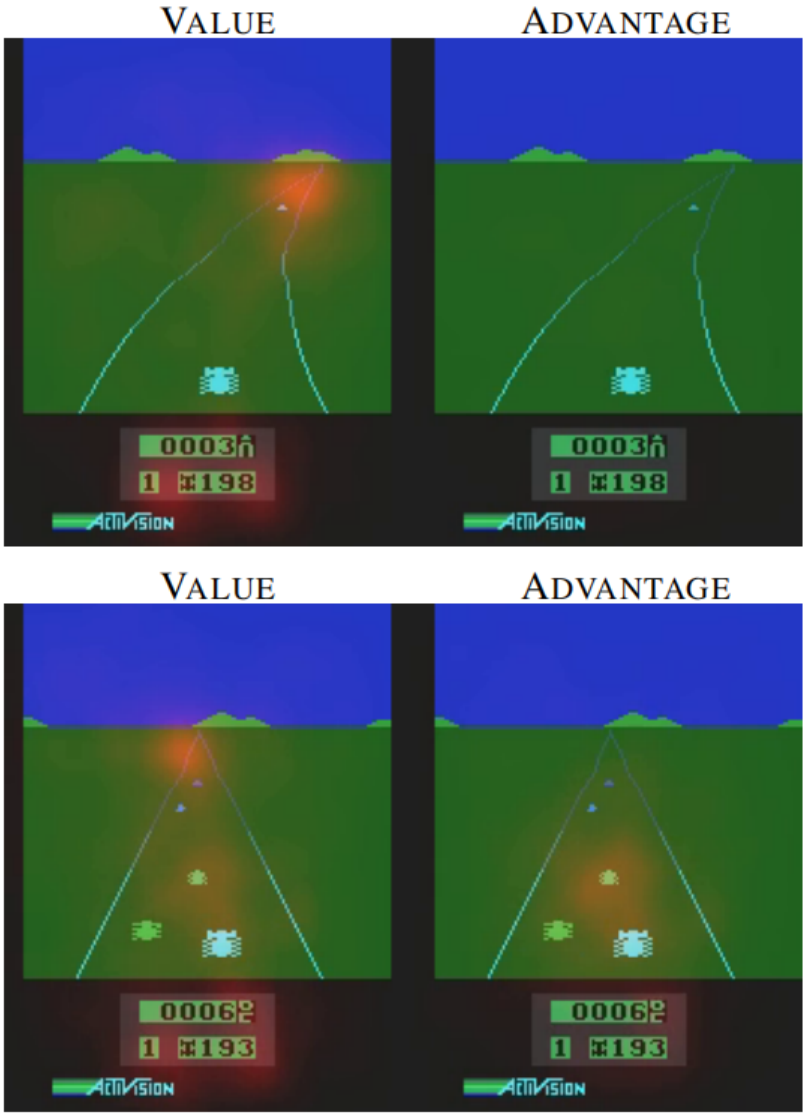


图 0.13　观看、参与和驾驶：在经过训练的对决架构下，雅达利游戏 Enduro 中的价值流与优势流显著性图 (红色标注覆盖层)。价值流学着关注道路状况；优势流则学着仅在其前方有车辆时才进行关注，从而避免碰撞。

性能提升。需要说明的是，采用相同架构和超参数的强化学习智能体，仅需通过观察图像像素和游戏得分，就能够玩 57 款不同游戏。实验结果表明，相较于 Mnih 等人[2] 和以及 van Hasselt 等人[10] 提出的单流基准模型，这种架构带来了显著的性能提升。将优先回放技术[11] 与提出的对决网络相结合，成为了这一热门领域新的前沿成果。

1.1 相关工作

关于保持独立的价值和优势函数的概念可以追溯到 Baird[12] 的著作。在 Baird 最初的优势更新算法中，共享的贝尔曼残差更新方程被分解为两个独立的更新过

程：一个用于状态价值函数，另一个用于和它相关的优势函数。研究表明，在简单的连续时间领域中，优势更新算法比 Q 学习收敛速度更快[13]。其后续发展中，优势学习算法，用于表示单一的优势函数[14]。

对决式架构通过单一深度模型同时实现价值函数 $V(s)$ 和优势 $A(s,a)$ 函数，其输出通过两者的融合生成状态动作价值 $Q(s,a)$ 。与优势更新方法不同，该架构在设计上实现了表征与算法的解耦。因此，这种对决式架构可与多种无模型强化学习算法结合使用。

优势函数在策略梯度中的应用由来已久，最早可追溯至[15]。作为该领域的最新成果，Schulman 等人[16] 通过在线估算优势值来降低策略梯度算法的方差。

在利用深度强化学习玩雅达利游戏领域，已有多个研究团队展开探索，包括 Mnih 等人[2]、Guo 等人[17]、Stadie 等人[18]、Nair 等人[6]、van Hasselt 等人[10]、Bellemare 等人[19] 以及 Schaul 团队[11]。Schaul 等人[11] 的研究成果是当前已发表的最先进水平。

2. 背景

我们研究的是一种序列化决策问题设定场景：智能体在离散时间步长中与环境 $\mathcal{E}$ 进行交互 (相关理论可参考 Sutton 与 Barto 于 1998 年提出的经典模型[20])。以雅达利游戏领域为例，智能体在时间步 t 会感知由 M 帧图像组成的视频序列 s_t : $s_t = (x_{t-M+1}, \ldots, x_t) \in \mathcal{S}$。随后，智能体从离散动作集合 $a_t \in \mathcal{A} = \{1, \ldots, |\mathcal{A}|\}$ 中选择动作，并通过游戏模拟器获取奖励信号 r_t。

智能体的目标是最大化期望的贴现收益，其中我们定义贴现收益为 $R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$ 。在此公式中，$\gamma \in [0,1]$ 是一个折扣因子，用于权衡即时奖励和未来奖励的重要性。

对于采用随机策略 π 的智能体，状态-动作对 (s,a) 和状态 s 的值定义如下：

$$Q^\pi(s,a) = \mathbb{E}\left[R_t \mid s_t = s, a_t = a, \pi\right] \text{且}$$
$$V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}\left[Q^\pi(s,a)\right]. \tag{1}$$

上文提到的状态-动作价值函数 (简称 Q 函数) 可以通过动态规划进行递归计算：

$$Q^\pi(s,a) = \mathbb{E}_{s'}\left[r + \gamma \mathbb{E}_{a'\sim\pi(s')}\left[Q^\pi(s',a')\right] \mid s, a, \pi\right].$$

我们定义最优 $Q^*(s,a) = \max_\pi Q^\pi(s,a)$。在确定性策略 $a = \arg\max_{a'\in\mathcal{A}} Q^*(s,a')$ 下，可推导出 $V^*(s) = \max_a Q^*(s,a)$。基于此，最优 Q 函数满足贝尔曼方程：

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]. \tag{2}$$

我们定义了另一个重要的量，即与价值函数和 Q 函数有关的优势函数：

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{3}$$

注意 $\mathbb{E}_{a\sim\pi(s)}\left[A^\pi(s,a)\right] = 0$。直观来说，价值函数 V 衡量的是处于特定状态 s 时的优劣程度。而 Q 函数则衡量当处于该状态时选择某个动作的价值。优势函数通过从 Q 函数中减去状态价值，来获得各动作的相对重要性量化指标。

补充：

$$\begin{aligned}
\mathbb{E}_{a\sim\pi(s)}\left[A^\pi(s,a)\right] &= \mathbb{E}_{a\sim\pi(s)}\left[Q^\pi(s,a) - V^\pi(s)\right] \\
&= \mathbb{E}_{a\sim\pi(s)}\left[Q^\pi(s,a)\right] - \mathbb{E}_{a\sim\pi(s)}\left[V^\pi(s)\right] \\
&= \sum_a \pi(a|s) Q^\pi(s,a) - \sum_a \pi(a|s) V^\pi(s) \\
&= V^\pi(s) - V^\pi(s) \cdot \sum_a \pi(a|s) \\
&= V^\pi(s) - V^\pi(s) \cdot 1 \\
&= 0.
\end{aligned}$$

2.1 深度 Q 网络

如前一节所述，价值函数是高维对象。为了对其进行近似，我们可以使用带有参数 θ 的 Q 网络：$Q(s,a;\theta)$。为了估计这个网络，我们在第 i 次迭代时优化以下损失函数序列：

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right], \tag{4}$$

其中

$$y_i^{DQN} = r + \gamma \max_{a'} Q\left(s',a';\theta^-\right), \tag{5}$$

其中 θ^- 表示一个固定且独立的目标网络中的参数。我们可以尝试使用标准 Q 学习方法在线学习该网络 $Q(s,a;\theta)$ 的参数。然而，这种估计器在实际应用中表现欠佳。Mnih 等人[2] 的关键创新在于：在固定迭代次数内冻结目标网络 $Q\left(s',a';\theta^-\right)$ 的参数，同时通过梯度下降法更新在线网络 $Q(s,a;\theta_i)$ 的参数 (这显著提升了算法的稳定性)。具体的梯度更新方式为：

$$\nabla_{\theta_i} L_i\left(\theta_i\right)=\mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN}-Q\left(s,a;\theta_i\right)\right)\nabla_{\theta_i} Q\left(s,a;\theta_i\right)\right].$$

这种方法是无模型的，因为状态和奖励是由环境产生的。它也是离轨策略，因为这些状态和奖励是通过与正在学习的在线策略不同的行为策略 (在 DQN 中是 ϵ-贪婪) 获得的。

深度 Q 网络 (DQN) 成功的另一个关键要素是经验回放[2,21]。在学习过程中，智能体会从许多幕中积累经验 $e_t=(s_t,a_t,r_t,s_{t+1})$，形成数据集 $\mathcal{D}_t=\{e_1,e_2,\cdots,e_t\}$。与传统时间差分学习仅使用当前经验不同，训练 Q 网络时需要从 $\mathcal{D}$ 中进行随机均匀采样，生成小批量经验值进行训练。因此损失序列的形式为：

$$L_i\left(\theta_i\right)=\mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}\left[\left(y_i^{DQN}-Q\left(s,a;\theta_i\right)\right)^2\right].$$

经验回放通过在多次更新中重复使用样本经验来提高数据效率，更重要的是，由于从回放缓冲区均匀采样减少了更新中使用的样本之间的相关性，减小了方差。

2.2 双深度 Q 网络

前一部分描述了 DQN 的主要组成，如 Mnih 等人所述[2]。在本文中，我们采用了 van Hasselt 等人提出的改进版双 DQN(Double DQN) 学习算法[10]。在 Q 学习和 DQN 中，最大值运算符 max 在选择和评估动作时使用相同数值，这可能导致价值估计过于乐观[22]。为解决这一问题，DDQN 采用了以下目标函数：

$$y_i^{DDQN}=r+\gamma Q(s',\arg\max_{a'}Q\left(s',a';\theta_i\right);\theta^-). \tag{6}$$

DDQN 与 DQN 相同[2]，但将目标 y_i^{DQN} 替换为 y_i^{DDQN} 。DDQN 的伪代码如附录 A 所示。

2.3 优先回放

近期，基于 DDQN，优先级经验回放领域的一项创新使得性能进一步提升[11]。其核心思路是提高那些具有高预期学习进步 (将绝对时序差值作为衡量指标) 的经验元组的回放概率。相较于传统均匀经验回放方法，这种方法不仅显著提升了学习速度，还在雅达利基准测试套件的多数游戏中实现了更优的最终策略质量。

为了强化我们关于对决架构与算法创新互补的说法，我们展示了它提高均匀和优先级回放基准实验的性能 (我们选择了更容易实现的基于排名的变体)，由此产生的优先级对决变体实现了新的最先进性能。

3 对决式网络架构

如图 2 所示，我们新架构的核心观点在于：对于许多状态而言，无需估算每个动作选择的具体价值。例如在 Enduro 游戏场景中，只有当碰撞即将发生时，才需要决定是左转还是右转。某些状态下，明确选择哪个动作至关重要，但在许多其他情况下，动作选择对后续结果并无影响。然而对于基于自举法的算法而言，每个状态的价值估计都至关重要。

为实现这一创新性见解，我们设计了一种单 Q-网络架构 (如图 1 所示)，并将其命名为“对决网络”。该网络的底层结构沿用了原始 DQN[2] 的卷积层设计。但与传统方法不同的是，我们没有在卷积层后简单堆砌全连接层，而是创新性地采用了两条独立的全连接层 (流)。这两条流经过专门设计，能够分别计算价值函数和优势函数的估计值。最终，两条流的输出会被整合生成统一的 Q 函数。与[2] 一样，网络的输出是一组 Q 值，每个动作对应一个值。

由于对决网络的输出是一个 Q 函数，因此可以使用许多现有的算法进行训练，如 DDQN 和 SARSA。此外，它还可以采用这些算法的任何改进形式，包括更好的回放记忆、更好的探索策略、内在动机等。

将两个全连接层流组合起来输出 Q 估计值的模块需要非常深思熟虑的设计。

根据优势表达式 $Q^{\pi}(s,a) = V^{\pi}(s) + A^{\pi}(s,a)$ 和状态价值表达式 $V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}\left[Q^{\pi}(s,a)\right]$ 可知，$\mathbb{E}_{a\sim\pi(s)}\left[A^{\pi}\left(s,a\right)\right] = 0$ 。此外，对于确定性策略，即 $a^{*} = \arg\max_{a'\in\mathcal{A}} Q\left(s,a'\right)$ ，可得 $Q\left(s,a^{*}\right) = V(s)$ ，因此 $A\left(s,a^{*}\right) = 0$。

我们考虑图 1 所示的对决网络架构：其中一条全连接层流输出标量值 $V(s;\theta,\beta)$，另一条全连接层流则输出 $|\mathcal{A}|$-维的向量 $A(s,a;\theta,\alpha)$。这里，θ 表示卷积层的参数，而 α 和 β 则是两条全连接层流的参数。

使用优势的定义，我们可能会去构建如下的聚合模块：

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha), \tag{7}$$

注意，此表达式适用于所有 (s,a) 实例；也就是说，若要以矩阵形式表达方程 (7)，我们需要复制标量 $V(s;\theta,\beta)$，$|\mathcal{A}|$ 次。

然而，我们必须牢记，$Q(s,a;\theta,\alpha,\beta)$ 只是真实 Q-函数的参数化估计。此外，若认为 $V(s;\theta,\beta)$ 是状态值函数的良好估计量，或认为 $A(s,a;\theta,\alpha)$ 能合理估计优势函数，这种结论都是不成立的。

方程 (7) 是不可辨识的，因为给定 Q 时无法唯一确定 V 和 A 。具体来说，若在 $V(s;\theta,\beta)$ 中添加一个常数，同时对 $A(s,a;\theta,\alpha)$ 做相同减法操作，这两个计算结果会相互抵消，最终得到相同的 Q 值。这种不可辨识性直接导致了该方程在实际

应用中表现欠佳。

为解决可辨识性问题，我们可以强制优势函数估计器在选定动作处保持零值。具体来说，我们让网络的最后一层模块实现前向映射

$$Q(s,a;\theta,\alpha,\beta)=V(s;\theta,\beta)+\left(A(s,a;\theta,\alpha)-\max_{a'\in|\mathcal{A}|}A\left(s,a';\theta,\alpha\right)\right). \tag{8}$$

现在，对于 $a^*=\arg\max_{a'\in\mathcal{A}}Q\left(s,a';\theta,\alpha,\beta\right)=\arg\max_{a'\in\mathcal{A}}A\left(s,a';\theta,\alpha\right)$，我们得到 $Q\left(s,a^*;\theta,\alpha,\beta\right)=V(s;\theta,\beta)$ 。因此，流 $V(s;\theta,\beta)$ 提供了价值函数的估计值，而另一条流则生成优势函数的估计值。

另一个替代模块用平均值替换最大值运算符 max：

$$Q(s,a;\theta,\alpha,\beta)=V(s;\theta,\beta)+\left(A(s,a;\theta,\alpha)-\frac{1}{|\mathcal{A}|}\sum_{a'}A\left(s,a';\theta,\alpha\right)\right). \tag{9}$$

一方面，这种处理方式会破坏公式 V 和 A 的原始语义，因为它们现在被一个常数干扰了；但另一方面，这也增强了优化过程的稳定性：通过公式 (9)，优势值只需随着均值变化而调整，而无需像公式 (8) 那样对最优动作的优势进行补偿性调整。我们还尝试过使用 softmax 版本的公式 (8)，但发现其效果与简化版公式 (9) 的模块相差无几。因此，本文所有实验均采用公式 (9) 的优化模块。

需要注意的是，虽然在公式 (9) 中减去均值有助于提高可辨识性，但它并不会改变 A(也包括 Q) 数值的相对顺序，因此仍能保持基于公式 (7) 中 Q 值的任何贪心或 ϵ-贪心策略。执行决策时，只需评估优势流即可做出判断。

需要特别说明的是，公式 (9) 应视为网络架构的组成部分而非独立算法步骤。与标准 Q 神经网络 (如 Mnih 等人[2] 提出的 Q- 深度网络) 类似，对决网络架构的训练仅需反向传播即可完成。估计值 $V(s;\theta\beta)$ 和 $A(s,a;\theta,\alpha)$ 的计算过程无需额外监督或算法调整，系统即可自动完成。

由于对决架构与标准 Q 网络共享相同的输入输出接口，我们可以使用 Q 网络 (例如，DDQN 和 SARSA) 的所有学习算法来训练对决架构。

4．实验

现在我们展示对决网络的实际性能。我们从一个简单的策略评估任务开始，然后展示更大规模的结果，学习通用的雅达利游戏策略。

4.1 策略评估

我们首先通过策略评估任务来测试对决架构的性能。为此，我们选择了特定的任务，用于评估网络架构，没有混淆因素，如探索策略的选择，以及策略改进和

策略评估之间的相互作用。

在本实验中，我们采用时间差分学习 (没有资格迹，即 $\lambda = 0$) 来学习 Q 值。具体来说，给定一个行为策略 π ，我们通过优化公式 (4) 中的成本序列来估计状态-动作价值 $Q^{\pi}(\cdot,\cdot)$，其目标为:

$$y_i = r + \gamma \mathbb{E}_{a' \sim \pi(s')} \left[Q\left(s', a'; \theta_i\right) \right].$$

上述更新规则与期望 SARSA [23] 的规则相同。但是，我们并没有像期望 SARSA 那样修改行为策略。

为评估学习到的 Q 值，我们选择了一个简单的环境模型。该模型中所有 $(s,a) \in \mathcal{S} \times \mathcal{A}$ 的精确 $Q^{\pi}(s,a)$ 值均可单独计算。这个被命名为“走廊” (corridor) 的环境由三条相连的走廊组成 (如图 3 所示)，智能体从环境左下角出发，必须移动到右上角才能获得最大奖励。系统共有 5 种动作：上下左右和空操作。我们还可以自由添加任意数量的空操作。在本实验设置中，两条垂直走廊各有 10 个状态，而水平走廊则有 50 个状态。

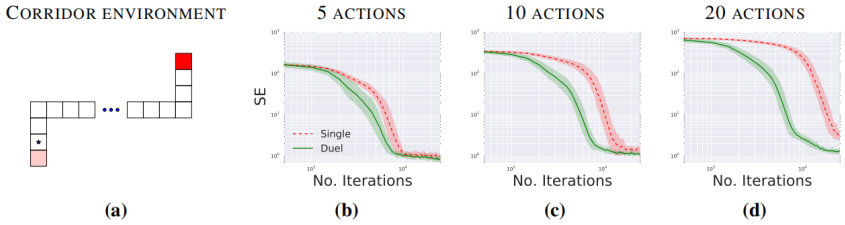


图 0.14　(a) 走廊环境。星号标记起始状态。状态的红色深浅表示智能体到达该状态时获得的奖励。当达到任一奖励状态时，游戏终止。智能体的动作包括向上、向下、向左、向右以及不采取任何动作。图 (b)、(c) 和 (d) 展示了在对数-对数坐标系下使用 5，10 和 20 个动作进行策略评估时的平方误差。对决网络 (Duel) 始终优于传统的单流网络 (Single)，且性能差距会随着动作数量的增加而增大。

我们使用一个 ϵ-贪婪策略作为行为策略 π，它以概率 ϵ 选择一个随机动作，或根据最优 Q 函数 $\arg\max_{a \in \mathcal{A}} Q^{*}(s,a)$ 以 $1-\epsilon$ 的概率选择一个动作。在我们的实验中，ϵ 被选为 0.001。

我们在包含 5，10 和 20 个动作的三种走廊环境变体上，对比了单流 Q 架构与对决架构的性能表现。其中 10 和 20 动作版本是通过在原始环境中添加空操作生成的。我们采用平方误差 (SE) 与真实状态值 $\sum_{s \in \mathcal{S}, a \in \mathcal{A}} (Q(s,a;\theta) - Q^{\pi}(s,a))^2$ 进行对比评估。单流架构采用三层多层感知机 (MLP)，每层包含 50 个隐藏单元。对决架构同样由三层组成，但其在首层 50 个单元的隐藏层之后，网络会分叉出两条流，每条均为包含 25 个隐藏单元的两层 MLP。对比结果如图 3 所示。

实验结果表明，在 5 个动作阶段，两种架构的收敛速度基本持平。但随着动作数量增加，对决网络架构的表现明显优于传统 Q-网络。在对决网络中，$V(s;\theta,\beta)$

流通过学习通用价值函数并将其共享给多个相似动作 s，从而提升了收敛速率。这一发现为后续研究指明了极具潜力的方向。由于许多动作空间较大的控制任务都具备这一特性，因此我们有理由相信，与传统的单流网络相比，对决网络往往能实现更快的收敛速度。在接下来的章节中，我们将看到对决网络在多种雅达利游戏中带来了显著的性能提升。

4.2 通用雅达利游戏玩法

我们在街机学习环境[24] 上对所提出的方法进行了全面评估。该环境包含 57 款雅达利游戏，其核心挑战在于：仅通过原始像素观测数据和游戏奖励信息，部署单一算法架构并固定超参数设置，使其能够学会玩所有游戏。该环境要求极高，不仅因为其包含大量高度多样化的游戏，还因其观测数据拥有高维特征。

我们严格遵循 van Hasselt 等人[10] 的实验设置，并通过单流 Q 网络与他们的结果进行对比。我们使用附录 A 中提出的 DDQN 算法训练对决网络。在本节结尾，我们引入了优先级经验回放[11]。

我们的网络架构沿用了 DQN[2,10] 的底层卷积结构。该架构包含 3 个卷积层和 2 个全连接层：首层卷积采用 32 个 8×8 滤波器，步长为 4；次层使用 64 个 4×4 滤波器，步长调整为 2；末层则配置 64 个 3×3 滤波器，步长保持 1。如图 1 所示，对决网络结构分为两个全连接层组成的流。价值流与优势流均包含一个 512 单元的全连接层，两者的最终隐藏层均为全连接结构。价值流输出单一结果，而优势流则具有独立的输出节点，输出的数量与有效动作数量相同[2]。我们通过公式 (9) 描述的模块将价值流与优势流进行整合。在所有相邻层之间插入整流器非线性单元[25]。

我们沿用了 van Hasselt 等人[10] 提出的优化器和超参数设置，但将学习率略微调低 (针对双 DQN 模型我们未作此调整，因其可能破坏其性能)。由于优势流和价值流在反向传播时都会将梯度传递至最后一层卷积层，我们对进入最后一层卷积层的组合后的梯度进行 $1/\sqrt{2}$ 的归一化处理。这种简单的优化策略能有效提升模型稳定性。此外，我们将梯度范数限制在 10 以内。虽然这种梯度裁剪方法在深度强化学习领域并非标准做法，但在循环神经网络训练中较为常见[26]。

为明确区分对决网络架构的贡献，我们采用与前述完全相同的流程，使用单流网络重新训练了 DDQN 模型。具体而言，我们在网络首层全连接层应用梯度裁剪技术，并采用 1024 个隐藏单元，使对决网络架构与单网络架构的参数量基本相当。我们将这个重新训练的模型命名为单剪枝 (Single Clip)，而 van Hasselt 等人[10] 训练的模型则称为原始模型 Single。

[2]在 ALE 环境中，操作的数量介于 3 到 18 次操作之间。

与 van Hasselt 等人[10] 的研究一致，我们通过最多 30 次无操作启动游戏，为智能体生成随机初始位置。为评估该方法，我们通过与人类专家算法和基准智能体的最优成绩对比，以百分比形式 (正负值) 衡量得分的提升幅度：

$$\frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Baseline}}}{\max\{\text{Score}_{\text{Human}}, \text{Score}_{\text{Baseline}}\} - \text{Score}_{\text{Random}}}. \tag{10}$$

我们对人类和基础智能体的分数取最大值，因为它可以防止不显著的变化出现。当智能体与基准模型表现均不理想时，仍能取得显著提升。例如，当基准模型仅达到人类水平的 1% 时，若某个智能体能达到 2% 的人类表现，不应简单视为其表现翻倍。我们选择不单纯以人类水平百分比作为衡量标准，因为某些游戏项目中相对于基准模型的微小差距，可能转化为人类表现差异的数百个百分点。

包含广泛内容的 57 款游戏的结果汇总在表 1 中。详细结果请参见附录。

通过这项 30 次无操作性能指标的评估可以明显看出，对决网络 (Duel Clip) 的表现远超同等容量的 Single Clip，也大幅优于 van Hasselt 等人[10] 提出的基准模型 (Single)。为便于比较，我们同时展示了 Mnih 团队[2] 提出的深度 Q 网络，即 Nature DQN(自然深度 Q 网络) 的测试结果。

图 4 显示了对决网络相对于 van Hasselt 等人[10] 提出的基线 Single 的改进。我们再次看到，改进往往是戏剧性的。

如表 1 所示，Single Clip 的表现优于 Single 。我们验证了这种增益主要来源于梯度裁剪。因此，我们在所有新方法中都引入了梯度裁剪。

表 1. 所有 57 个雅达利游戏的平均分和中位数，以人类表现的百分比衡量。

	30 no-ops		**Human Starts**	
	Mean	**Median**	**Mean**	**Median**
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%	219.6%	68.5%

在 57 款游戏中，Duel Clip 以 75.4% 的游戏表现优于 Single Clip(43/57)。在 80.7% 的游戏 (46/57) 中，其得分也高于 Single。在包含 18 种动作的游戏里，Duel Clip 以 26/30 的占比高出 86.6%，这与前文结论一致。总体而言，我们的智能体 (Duel Clip) 在 57 款游戏中有 42 款达到了人类水平的表现。所有游戏的原始分数，以及人类表现百分比的测量结果，都在附录中呈现。

对人类开局的鲁棒性　30 项无操作指标的一个不足之处在于，智能体未必能

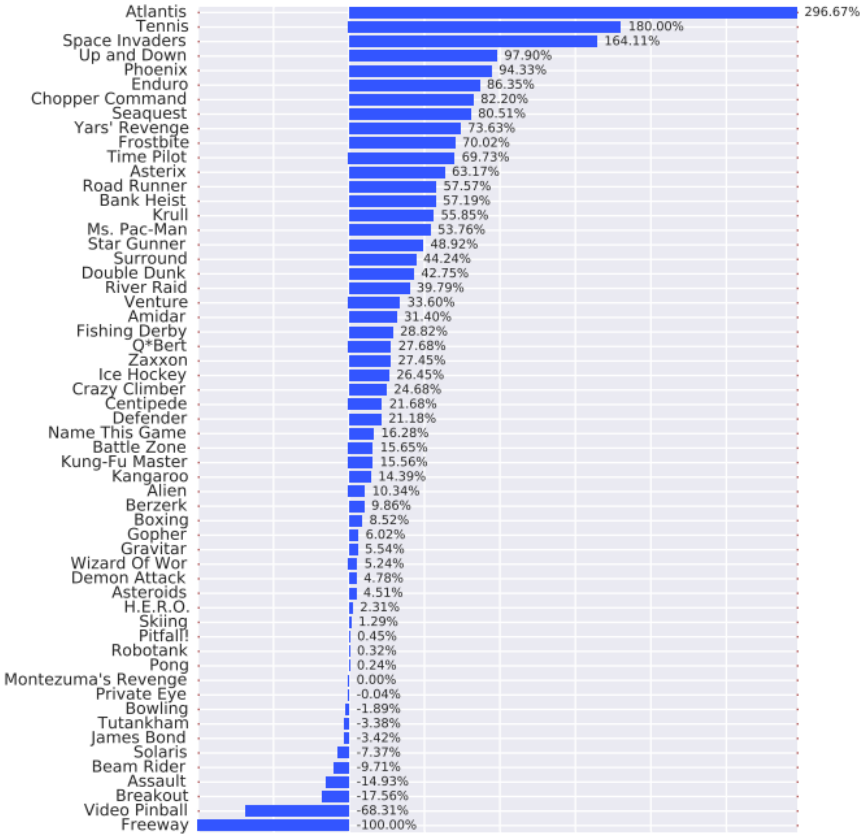


图 0.15　基于公式 (10) 所述指标，与 van Hasselt 等人[10] 提出的单网络架构相比，对决式网络的改进。右侧柱状图显示对决式网络相较于单流网络的性能提升幅度。

很好地泛化以应对雅达利游戏。由于雅达利环境的确定性特征，从一个独特的开局开始，智能体只需记住动作序列就能学会取得良好表现。

为获得更鲁棒的评估指标，我们采用了 Nair 等人[6] 提出的方法论。具体而言，在每局游戏中，我们从人类专家的轨迹中选取 100 个起始点。针对每个起始点，我们会启动长达 108,000 帧的评估幕。智能体的表现仅基于该起始点之后累积的奖励进行评估。我们将这种评估指标称为 Human Starts。

如表 1 所示，在人类起始指标下，Duel Clip 再次超越单流变体。具体而言，我们的智能体在 70.2%(57 局中的 40 局) 的游戏中表现优于 Single；而在包含 18 个动作的游戏场景中，Duel Clip 更是以 83.3% 的胜率 (30 局中的 25 局) 占据优势。

与优先级经验回放技术结合　对决架构可轻松与其他算法改进版本相结合。特别值得注意的是，优先经验回放已被证实能显著提升雅达利游戏的性能[11]。此外，由于优先级与对决架构针对学习过程的不同维度，二者结合展现出巨大潜力。因此在我们的最终实验中，我们重点研究了对决架构与优先经验回放的整合方案。实验采用 DDQN 算法的优先级变体 (Prior．Single) 作为新基准算法，该算法替代了

传统经验均匀采样方法，通过基于优先级的采样方法实现元组排序。我们保留了文献[11]所述的优先级重放缓冲区所有参数，具体包括：优先级指数设为 0.7，重要性采样指数采用从 0.5 到 1 的退火式衰减方案。我们将该基线方法与上述对决网络架构相结合，并继续应用梯度裁剪技术 (Prior. Duel Clip)。

需要注意的是，尽管这些扩展机制 (优先级排序、对决和梯度裁剪) 在目标上相互独立，但它们之间存在着微妙的交互作用。例如，优先级排序与梯度裁剪会产生联动效应。当采样过程中出现绝对时序差异常高的情况时，往往会生成范数更大的梯度。为避免这种负面交互，我们在含 9 种游戏的子集中对学习率和梯度裁剪范数进行了初步调整。经过粗调后，最终确定的学习率参数为 6.25×10^{-5}，梯度裁剪范数参数仍保持前一节设定的 10。

在对全部 57 款雅达利游戏进行评估时，我们设计的优先级对决智能体表现显著优于优先级基准智能体和单独对决智能体。表 1 展示了该智能体相对于人类表现的平均分与中位数百分比。当使用最多 30 次无操作动作初始化游戏时，其平均得分达到 591%，中位数得分则为 172%。图 5 通过公式 (10) 所述指标，直观呈现了优先级基准智能体与优先级对决版本之间的直接对比结果。

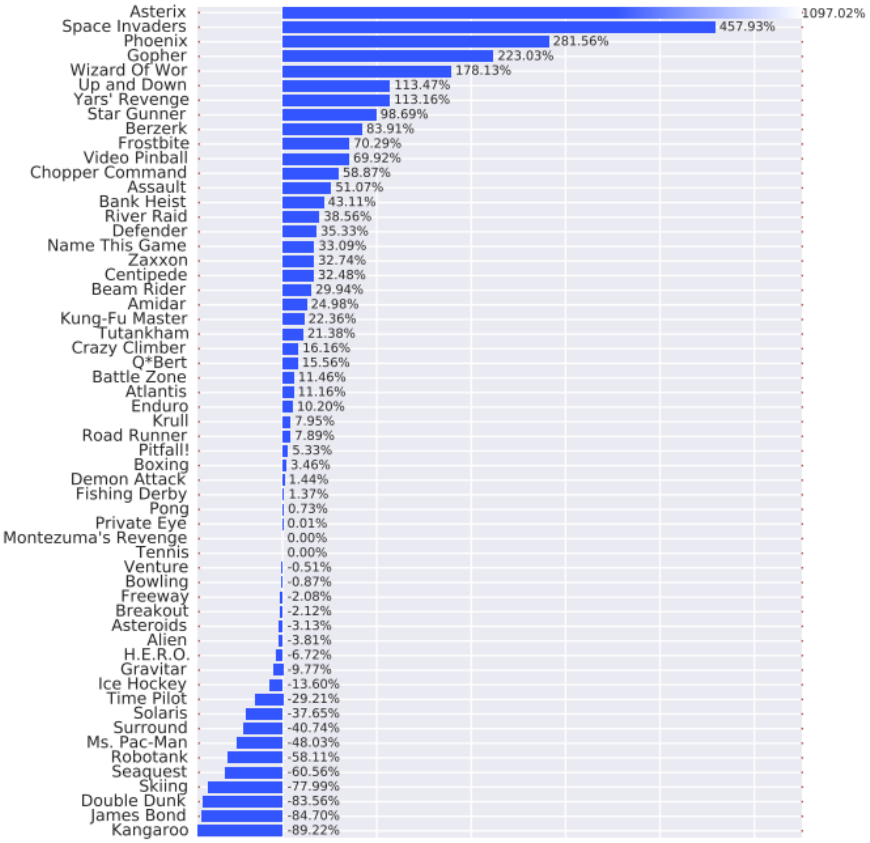


图 0.16　使用与图 4 相同的度量标准，展示了对决架构相较于优先级 DDQN 基线的改进。同样地，在大多数游戏中，对决架构相比单流基线有了显著提升。

优先回放和对决网络的结合在流行的 ALE 环境中相较于以前最先进的结果有了巨大的改进。

显著性图 为了更好地理解价值流与优势流的作用机制，我们计算了显著性图[9]。具体而言，为可视化价值流所感知的图像关键区域，我们计算了 $\widehat{V}$ 关于输入帧的雅可比矩阵绝对值 $\left|\nabla_s \widehat{V}(s;\theta)\right|$ 。同理，为可视化优势流所感知的图像关键区域，我们计算 $\left|\nabla_s \widehat{A}(s, \arg\max_{a'} \widehat{A}(s,a');\theta))\right|$。这两个量与输入帧具有相同的维度，因此可以很容易地与输入帧一起可视化。

我们将灰度输入帧置于绿色通道和蓝色通道，显著性图则显示在红色通道。这三个通道共同构成 RGB 图像。图 2 展示了 Enduro 游戏中两个不同时间步长下的价值流与优势流显著性图。正如引言所述，价值流关注可能影响未来表现的车辆出现位置，同时也会关注得分。而优势流则更关注那些处于即时碰撞路径上的车辆。

5. 讨论

对决网络架构的优势部分源于其高效学习状态价值函数的能力。在对决网络架构中，每次更新 Q 值时，价值流 V 也会同步更新，这与单流架构中仅更新单一动作值、其他所有动作值保持不变的更新方式形成鲜明对比。这种更频繁的价值流更新为 V 分配了更多资源，从而能更精准地逼近状态价值。对于 Q 学习[20] 这类基于时间差分的方法而言，这种精度是其有效运行的前提。实验结果也印证了这一现象：当动作数量较多时，对决网络架构相较于单流 Q 网络的优势愈发显著。

此外，特定状态下不同动作的 Q 值差异通常相对于 Q 值本身来说微乎其微。例如，在 Seaquest 游戏中使用 DDQN 进行训练后，各访问状态的平均动作间隔 (即特定状态下最优动作与次优动作的 Q 值差值) 约为 0.4，而这些状态的平均状态值却高达 15。这种量级差异可能导致微小的噪声干扰，进而引发动作顺序重排，最终导致原本近似贪婪的策略突然切换。具有独立优势流的对决架构对这种影响是鲁棒的。

6. 结论

我们提出了一种新型神经网络架构，在深度 Q 网络中实现了价值与优势的解耦，同时共享特征学习模块。这种创新的对决式架构结合多项算法优化，在极具挑战性的雅达利游戏领域中，相较现有深度强化学习方法展现出显著优势。本文展示的研究成果，标志着该热门领域达到了新的技术前沿水平。

参考文献

[1] LeCun Y., Bengio Y., and Hinton G. Deep learning. Nature, 521(7553):436–444, 2015.

[2] Mnih V., Kavukcuoglu K., Silver D., and et al. Human-level control through deep reinforcement learning. Nature, 518:529–533, 2015.

[3] Levine S., Finn C., Darrell T., and Abbeel P. End-to-end training of deep visuomotor policies. Journal of Machine Learning Research, 17:1–40, 2015.

[4] Ba J., Mnih V., and Kavukcuoglu K. Multiple object recognition with visual attention. ICLR, 2015.

[5] Watter M., Springenberg J., Boedecker J., and Riedmiller M. Embed to control: A locally linear latent dynamics model for control from raw images. In NIPS, 2015.

[6] Nair A., Srinivasan P., Blackwell S., and et al. Massively parallel methods for deep reinforcement learning. In Deep Learning Workshop, ICML, 2015.

[7] Maddison C. J., Huang A., Sutskever I., and Silver D. Move evaluation in Go using deep convolutional neural networks. ICLR, 2015.

[8] Silver D., Huang A., Maddison C. J., and et al. Mastering the game of Go with deep neural networks and tree search. Nature, 529:484–489, 2016.

[9] Simonyan K., Vedaldi A., and Zisserman A. Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint, arXiv:1312.6034, 2013.

[10] van Hasselt H., Guez A., and Silver D. Deep reinforcement learning with double Q-learning. arXiv preprint, arXiv:1509.06461, 2015.

[11] Schaul T., Quan J., Antonoglou I., and Silver D. Prioritized experience replay. In ICLR, 2016.

[12] Baird L. C. Advantage updating. Technical report, Wright Laboratory, WL-AFOSR-93-0063, 1993.

[13] Harmon M. E., Baird L. C., and Klopf A. H. Advantage updating applied to a differential game. In NIPS, 1995.

[14] Harmon M. E., and Baird L. C. Multi-player residual advantage learning with general function approximation. Technical report, Wright Laboratory, WL-AFOSR-96-1, 1996.

[15] Sutton R., McAllester D., Singh S., and Mansour Y. Policy gradient methods for reinforcement learning with function approximation. In NIPS, pp.1057-1063, 2000.

[16] Schulman J., Moritz P., Levine S., Jordan M., and Abbeel P. High-dimensional continuous control using generalized advantage estimation. arXiv preprint, arXiv:1506.02438, 2015.

[17] Guo X., Singh S., Lee H., Lewis R., and Wang X. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In NIPS, pp.3338-3346, 2014.

[18] Stadie B., Levine S., and Abbeel P. Incentivizing exploration in reinforcement learning with deep predictive models. arXiv preprint, arXiv:1507.00814, 2015.

[19] Bellemare M., Ostrovski G., Guez A., Thomas P., and Munos R. Increasing the action gap: New operators for reinforcement learning. In AAAI, 2016. To appear.

[20] Sutton R. S. and Barto A. G. Introduction to reinforcement learning. MIT Press, 1998.

[21] Lin L. J. Reinforcement learning for robots using neural networks. PhD thesis, Carnegie Mellon University, 1993.

[22] van Hasselt H. Double Q-learning. NIPS, 23:2613-2621, 2010.

[23] van Seijen H., van Hasselt H., Whiteson S., and Wiering M. A theoretical and empirical analysis of Expected Sarsa. In IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, pp. 177-184, 2009.

[24] Bellemare M., Naddaf Y., Veness J., and Bowling M. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253−279, 2013.

[25] Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36:193-202, 1980.

[26] Bengio Y., Boulanger-Lewandowski N., and Pascanu R. Advances in optimizing recurrent networks. In ICASSP, pp. 8624-8628, 2013.

A. 双重 DQN 算法

算法 2 带有对决网络的 DQN 结构及双重 Q-学习

输入: $\mathcal{D}$ -空的回放缓冲区；θ -初始网络参数；θ^- -θ 的拷贝
输入: N_r -回放缓冲区最大容量；N_b -训练批量大小；N^- -目标网络替换频率
for 每一幕 $e \in \{1, 2, \ldots, M\}$ **do**
    初始化帧序列 $\mathbf{x} \leftarrow ()$
    for $t \in \{0, 1, \ldots\}$ **do**
        设状态 $s \leftarrow \mathbf{x}$，根据策略 π_B 采样动作 a
        给定 (s, a)，从环境 $\mathcal{E}$ 中采样下一帧 x^t，并接收奖励 r，将 x^t 添加到 $\mathbf{x}$
        if $|\mathbf{x}| > N_f$ **then**
            从 $\mathbf{x}$ 中删除最旧的帧 $x_{t_{\min}}$
        end if
        设 $s' \leftarrow \mathbf{x}$，并将转移元组 (s, a, r, s') 添加到 $\mathcal{D}$，若 $|\mathcal{D}| \geq N_r$，则替换最旧的元组
        从 $\mathcal{D}$ 中均匀采样一个大小为 N_b 的小批量样本 (s, a, r, s')
        为 N_b 个元组中的每一项构造目标值:
        定义 $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$
        $$y_j = \begin{cases} r, & \text{若 } s' \text{ 是终止状态} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{否则} \end{cases}$$
        执行一次梯度下降更新，损失函数为 $\|y_j - Q(s, a; \theta)\|^2$
        每 N^- 步更新目标参数 $\theta^- \leftarrow \theta$
    end for
end for

表 2. 所有游戏的原始得分 (**30** 次无操作启动).

GAMES	NO. ACTIONS	RANDOM	HUMAN	DQN	DDQN	DUEL	PRIOR.	PRIOR. DUEL.
Alien	18	227.8	7,127.7	1,620.0	3,747.7	**4,461.4**	4,203.8	3,941.0
Amidar	10	5.8	1,719.5	978.0	1,793.3	**2,354.5**	1,838.9	2,296.8
Assault	7	222.4	742.0	4,280.4	5,393.2	4,621.0	7,672.1	**11,477.0**
Asterix	9	210.0	8,503.3	4,359.0	17,356.5	28,188.0	31,527.0	**375,080.0**
Asteroids	14	719.1	47,388.7	1,364.5	734.7	**2,837.7**	2,654.3	1,192.7
Atlantis	4	12,850.0	29,028.1	279,987.0	106,056.0	382,572.0	357,324.0	**395,762.0**
Bank Heist	18	14.2	753.1	455.0	1,030.6	**1,611.9**	1,054.6	1,503.1
Battle Zone	18	2,360.0	37,187.5	29,900.0	31,700.0	**37,150.0**	31,530.0	35,520.0
Beam Rider	9	363.9	16,926.5	8,627.5	13,772.8	12,164.0	23,384.2	**30,276.5**
Berzerk	18	123.7	2,630.4	585.6	1,225.4	1,472.6	1,305.6	**3,409.0**
Bowling	6	23.1	160.7	50.4	**68.1**	65.5	47.9	46.7
Boxing	18	0.1	12.1	88.0	91.6	**99.4**	95.6	98.9
Breakout	4	1.7	30.5	385.5	**418.5**	345.3	373.9	366.0
Centipede	18	2,090.9	12,017.0	4,657.7	5,409.4	7,561.4	4,463.2	**7,687.5**
Chopper Command	18	811.0	7,387.8	6,126.0	5,809.0	11,215.0	8,600.0	**13,185.0**
Crazy Climber	9	10,780.5	35,829.4	110,763.0	117,282.0	143,570.0	141,161.0	**162,224.0**
Defender	18	2,874.5	18,688.9	23,633.0	35,338.5	**42,214.0**	31,286.5	41,324.5
Demon Attack	6	152.1	1,971.0	12,149.4	58,044.2	60,813.3	71,846.4	**72,878.6**
Double Dunk	18	-18.6	-16.4	-6.6	-5.5	0.1	**18.5**	-12.5
Enduro	9	0.0	860.5	729.0	1,211.8	2,258.2	2,093.0	**2,306.4**
Fishing Derby	18	-91.7	-38.7	-4.9	15.5	**46.4**	39.5	41.3
Freeway	3	0.0	29.6	30.8	33.3	0.0	**33.7**	33.0
Frostbite	18	65.2	4,334.7	797.4	1,683.3	4,672.8	4,380.1	**7,413.0**
Gopher	8	257.6	2,412.5	8,777.4	14,840.8	15,718.4	32,487.2	**104,368.2**
Gravitar	18	173.0	3,351.4	473.0	412.0	**588.0**	548.5	238.0
H.E.R.O.	18	1,027.0	30,826.4	20,437.8	20,130.2	20,818.2	**23,037.7**	21,036.5
Ice Hockey	18	-11.2	0.9	-1.9	-2.7	0.5	**1.3**	-0.4
James Bond	18	29.0	302.8	768.5	1,358.0	1,312.5	**5,148.0**	812.0
Kangaroo	18	52.0	3,035.0	7,259.0	12,992.0	14,854.0	**16,200.0**	1,792.0
Krull	18	1,598.0	2,665.5	8,422.3	7,920.5	**11,451.9**	9,728.0	10,374.4
Kung-Fu Master	14	258.5	22,736.3	26,059.0	29,710.0	34,294.0	39,581.0	**48,375.0**
Montezuma's Revenge	18	0.0	4,753.3	**0.0**	0.0	0.0	0.0	0.0
Ms. Pac-Man	9	307.3	6,951.6	3,085.6	2,711.4	6,283.5	6,518.7	3,327.3
Name This Game	6	2,292.3	8,049.0	8,207.8	10,616.0	11,971.1	12,270.5	**15,572.5**
Phoenix	8	761.4	7,242.6	8,485.2	12,252.5	23,092.2	18,992.7	**70,324.3**
Pitfall!	18	-229.4	6,463.7	-286.1	-29.9	**0.0**	-356.5	0.0
Pong	3	-20.7	14.6	19.5	20.9	**21.0**	20.6	20.9
Private Eye	18	24.9	69,571.3	146.7	129.7	103.0	200.0	**206.0**
Q*Bert	6	163.9	13,455.0	13,117.3	15,088.5	**19,220.3**	16,256.5	18,760.3
River Raid	18	1,338.5	17,118.0	7,377.6	14,884.5	**21,162.6**	14,522.3	20,607.6
Road Runner	18	11.5	7,845.0	39,544.0	44,127.0	**69,524.0**	57,608.0	62,151.0
Robotank	18	2.2	11.9	63.9	65.1	**65.3**	62.6	27.5
Seaquest	18	68.4	42,054.7	5,860.6	16,452.7	**50,254.2**	26,357.8	931.6
Skiing	3	-17,098.1	-4,336.9	-13,062.3	-9,021.8	**-8,857.4**	-9,996.9	-19,949.9
Solaris	18	1,236.3	12,326.7	3,482.8	3,067.8	2,250.8	**4,309.0**	133.4
Space Invaders	6	148.0	1,668.7	1,692.3	2,525.5	6,427.3	2,865.8	**15,311.5**
Star Gunner	18	664.0	10,250.0	54,282.0	60,142.0	89,238.0	63,302.0	**125,117.0**
Surround	5	-10.0	6.5	-5.6	-2.9	4.4	**8.9**	1.2
Tennis	18	-23.8	-8.3	**12.2**	-22.8	5.1	0.0	0.0
Time Pilot	10	3,568.0	5,229.2	4,870.0	8,339.0	**11,666.0**	9,197.0	7,553.0
Tutankham	8	11.4	167.6	68.1	218.4	211.4	204.6	**245.9**
Up and Down	6	533.4	11,693.2	9,989.9	22,972.2	**44,939.6**	16,154.1	33,879.1
Venture	18	0.0	1,187.5	163.0	98.0	**497.0**	54.0	48.0
Video Pinball	9	16,256.9	17,667.9	196,760.4	309,941.9	98,209.5	282,007.3	**479,197.0**
Wizard Of Wor	10	563.5	4,756.5	2,704.0	7,492.0	7,855.0	4,802.0	**12,352.0**
Yars' Revenge	18	3,092.9	54,576.9	18,098.9	11,712.6	49,622.1	11,357.0	**69,618.1**
Zaxxon	18	32.5	9,173.3	5,363.0	10,163.0	12,944.0	10,469.0	**13,886.0**

表 3. 所有游戏的原始得分 (人类开局).

GAMES	NO. ACTIONS	RANDOM	HUMAN	DQN	DDQN	DUEL	PRIOR.	PRIOR. DUEL.
Alien	18	128.3	6,371.3	634.0	1,033.4	**1,486.5**	1,334.7	823.7
Amidar	10	11.8	1,540.4	178.4	169.1	172.7	129.1	**238.4**
Assault	7	166.9	628.9	3,489.3	6,060.8	3,994.8	6,548.9	**10,950.6**
Asterix	9	164.5	7,536.0	3,170.5	16,837.0	15,840.0	22,484.5	**364,200.0**
Asteroids	14	871.3	36,517.3	1,458.7	1,193.2	**2,035.4**	1,745.1	1,021.9
Atlantis	4	13,463.0	26,575.0	292,491.0	319,688.0	**445,360.0**	330,647.0	423,252.0
Bank Heist	18	21.7	644.5	312.7	886.0	**1,129.3**	876.6	1,004.6
Battle Zone	18	3,560.0	33,030.0	23,750.0	24,740.0	**31,320.0**	25,520.0	30,650.0
Beam Rider	9	254.6	14,961.0	9,743.2	17,417.2	14,591.3	31,181.3	**37,412.2**
Berzerk	18	196.1	2,237.5	493.4	1,011.1	910.6	865.9	**2,178.6**
Bowling	6	35.2	146.5	56.5	**69.6**	65.7	52.0	50.4
Boxing	18	-1.5	9.6	70.3	73.5	77.3	72.3	**79.2**
Breakout	4	1.6	27.9	354.5	368.9	**411.6**	343.0	354.6
Centipede	18	1,925.5	10,321.9	3,973.9	3,853.5	4,881.0	3,489.1	**5,570.2**
Chopper Command	18	644.0	8,930.0	5,017.0	3,495.0	3,784.0	4,635.0	**8,058.0**
Crazy Climber	9	9,337.0	32,667.0	98,128.0	113,782.0	124,566.0	127,512.0	**127,853.0**
Defender	18	1,965.5	14,296.0	15,917.5	27,510.0	33,996.0	23,666.5	**34,415.0**
Demon Attack	6	208.3	3,442.8	12,550.7	69,803.4	56,322.8	61,277.5	**73,371.3**
Double Dunk	18	-16.0	-14.4	-6.0	-0.3	-0.8	**16.0**	-10.7
Enduro	9	-81.8	740.2	626.7	1,216.6	2,077.4	1,831.0	**2,223.9**
Fishing Derby	18	-77.1	5.1	-1.6	3.2	-4.1	9.8	**17.0**
Freeway	3	0.1	25.6	26.9	28.8	0.2	**28.9**	28.2
Frostbite	18	66.4	4,202.8	496.1	1,448.1	2,332.4	3,510.0	**4,038.4**
Gopher	8	250.0	2,311.0	8,190.4	15,253.0	20,051.4	34,858.8	**105,148.4**
Gravitar	18	245.5	3,116.0	**298.0**	200.5	297.0	269.5	167.0
H.E.R.O.	18	1,580.3	25,839.4	14,992.9	14,892.5	15,207.9	**20,889.9**	15,459.2
Ice Hockey	18	-9.7	0.5	-1.6	-2.5	-1.3	-0.2	**0.5**
James Bond	18	33.5	368.5	697.5	573.0	835.5	**3,961.0**	585.0
Kangaroo	18	100.0	2,739.0	4,496.0	11,204.0	10,334.0	**12,185.0**	861.0
Krull	18	1,151.9	2,109.1	6,206.0	6,796.1	**8,051.6**	6,872.8	7,658.6
Kung-Fu Master	14	304.0	20,786.8	20,882.0	30,207.0	24,288.0	31,676.0	**37,484.0**
Montezuma's Revenge	18	25.0	4,182.0	47.0	42.0	22.0	**51.0**	24.0
Ms. Pac-Man	9	197.8	15,375.0	1,092.3	1,241.3	**2,250.6**	1,865.9	1,007.8
Name This Game	6	1,747.8	6,796.0	6,738.8	8,960.3	11,185.1	10,497.6	**13,637.9**
Phoenix	8	1,134.4	6,686.2	7,484.8	12,366.5	20,410.5	16,903.6	**63,597.0**
Pitfall!	18	-348.8	5,998.9	-113.2	-186.7	**-46.9**	-427.0	-243.6
Pong	3	-18.0	15.5	18.0	**19.1**	18.8	18.9	18.4
Private Eye	18	662.8	64,169.1	207.9	-575.5	292.6	670.7	**1,277.6**
Q*Bert	6	183.0	12,085.0	9,271.5	11,020.8	**14,175.8**	9,944.0	14,063.0
River Raid	18	588.3	14,382.2	4,748.5	10,838.4	**16,569.4**	11,807.2	16,496.8
Road Runner	18	200.0	6,878.0	35,215.0	43,156.0	**58,549.0**	52,264.0	54,630.0
Robotank	18	2.4	8.9	58.7	59.1	**62.0**	56.2	24.7
Seaquest	18	215.5	40,425.8	4,216.7	14,498.0	**37,361.6**	25,463.7	1,431.2
Skiing	3	-15,287.4	-3,686.6	-12,142.1	-11,490.4	-11,928.0	**-10,169.1**	-18,955.8
Solaris	18	2,047.2	11,032.6	1,295.4	810.0	1,768.4	**2,272.8**	280.6
Space Invaders	6	182.6	1,464.9	1,293.8	2,628.7	5,993.1	3,912.1	**8,978.0**
Star Gunner	18	697.0	9,528.0	52,970.0	58,365.0	90,804.0	61,582.0	**127073.0**
Surround	5	-9.7	5.4	-6.0	1.9	4.0	**5.9**	-0.2
Tennis	18	-21.4	-6.7	**11.1**	-7.8	4.4	-5.3	-13.2
Time Pilot	10	3,273.0	5,650.0	4,786.0	**6,608.0**	6,601.0	5,963.0	4,871.0
Tutankham	8	12.7	138.3	45.6	92.2	48.0	56.9	**108.6**
Up and Down	6	707.2	9,896.1	8,038.5	19,086.9	**24,759.2**	12,157.4	22,681.3
Venture	18	18.0	1,039.0	136.0	21.0	**200.0**	94.0	29.0
Video Pinball	9	20,452.0	15,641.1	154,414.1	367,823.7	110,976.2	295,972.8	**447,408.6**
Wizard Of Wor	10	804.0	4,556.0	1,609.0	6,201.0	7,054.0	5,727.0	**10,471.0**
Yars' Revenge	18	1,476.9	47,135.2	4,577.5	6,270.6	25,976.5	4,687.4	**58,145.9**
Zaxxon	18	475.0	8,443.0	4,412.0	8,593.0	10,164.0	9,474.0	**11,320.0**

表 4. 所有游戏的标准化得分 (**30 次无操作启动**) .

GAMES	DQN	DDQN	DUEL	PRIOR.	PRIOR. DUEL.
Alien	20.2%	51.0%	**61.4%**	57.6%	53.8%
Amidar	56.7%	104.3%	**137.1%**	107.0%	133.7%
Assault	781.0%	995.1%	846.5%	1433.7%	**2166.0%**
Asterix	50.0%	206.8%	337.4%	377.6%	**4520.1%**
Asteroids	1.4%	0.0%	**4.5%**	4.1%	1.0%
Atlantis	1651.2%	576.1%	2285.3%	2129.3%	**2366.9%**
Bank Heist	59.7%	137.6%	**216.2%**	140.8%	201.5%
Battle Zone	79.1%	84.2%	**99.9%**	83.8%	95.2%
Beam Rider	49.9%	81.0%	71.2%	139.0%	**180.6%**
Berzerk	18.4%	44.0%	53.8%	47.2%	**131.1%**
Bowling	19.8%	**32.7%**	30.8%	18.0%	17.1%
Boxing	732.5%	762.1%	**827.1%**	795.5%	823.1%
Breakout	1334.5%	**1449.2%**	1194.5%	1294.3%	1266.6%
Centipede	25.9%	33.4%	55.1%	23.9%	**56.4%**
Chopper Command	80.8%	76.0%	158.2%	118.4%	**188.1%**
Crazy Climber	399.1%	425.2%	530.1%	520.5%	**604.6%**
Defender	131.3%	205.3%	**248.8%**	179.7%	243.1%
Demon Attack	659.6%	3182.8%	3335.0%	3941.6%	**3998.3%**
Double Dunk	557.7%	607.9%	866.5%	**1723.3%**	280.5%
Enduro	84.7%	140.8%	262.4%	243.2%	**268.0%**
Fishing Derby	163.8%	202.4%	**260.7%**	247.7%	251.1%
Freeway	104.0%	112.5%	0.1%	**114.0%**	111.3%
Frostbite	17.1%	37.9%	107.9%	101.1%	**172.1%**
Gopher	395.4%	676.7%	717.5%	1495.6%	**4831.3%**
Gravitar	9.4%	7.5%	**13.1%**	11.8%	2.0%
H.E.R.O.	65.1%	64.1%	66.4%	**73.9%**	67.1%
Ice Hockey	76.9%	70.0%	96.4%	**103.2%**	89.6%
James Bond	270.1%	485.4%	468.8%	**1869.8%**	286.0%
Kangaroo	241.6%	433.8%	496.2%	**541.3%**	58.3%
Krull	639.3%	592.3%	**923.1%**	761.6%	822.2%
Kung-Fu Master	114.8%	131.0%	151.4%	174.9%	**214.1%**
Montezuma's Revenge	**0.0%**	0.0%	0.0%	0.0%	0.0%
Ms. Pac-Man	41.8%	36.2%	89.9%	**93.5%**	45.5%
Name This Game	102.8%	144.6%	168.1%	173.3%	**230.7%**
Phoenix	119.2%	177.3%	344.5%	281.3%	**1073.3%**
Pitfall!	-0.8%	3.0%	**3.4%**	-1.9%	3.4%
Pong	114.0%	117.8%	**118.2%**	117.1%	118.0%
Private Eye	0.2%	0.2%	0.1%	**0.3%**	0.3%
Q*Bert	97.5%	112.3%	**143.4%**	121.1%	139.9%
River Raid	38.3%	85.8%	**125.6%**	83.6%	122.1%
Road Runner	504.7%	563.2%	**887.4%**	735.3%	793.3%
Robotank	631.5%	643.7%	**645.1%**	617.5%	259.5%
Seaquest	13.8%	39.0%	**119.5%**	62.6%	2.1%
Skiing	31.6%	63.3%	**64.6%**	55.6%	-22.3%
Solaris	20.3%	16.5%	9.1%	**27.7%**	-9.9%
Space Invaders	101.6%	156.3%	412.9%	178.7%	**997.2%**
Star Gunner	559.3%	620.5%	924.0%	653.4%	**1298.3%**
Surround	26.5%	43.2%	86.9%	**114.6%**	67.6%
Tennis	**231.3%**	6.8%	186.2%	153.2%	153.2%
Time Pilot	78.4%	287.2%	**487.5%**	338.9%	239.9%
Tutankham	36.3%	132.5%	128.1%	123.7%	**150.1%**
Up and Down	84.7%	201.1%	**397.9%**	140.0%	298.8%
Venture	13.7%	8.3%	**41.9%**	4.5%	4.0%
Video Pinball	1113.7%	1754.3%	555.9%	1596.2%	**2712.2%**
Wizard Of Wor	51.0%	165.2%	173.9%	101.1%	**281.1%**
Yars' Revenge	29.1%	16.7%	90.4%	16.1%	**129.2%**
Zaxxon	58.3%	110.8%	141.3%	114.2%	**151.6%**

表 5. 所有游戏的标准化得分 (人类开局).

GAMES	DQN	DDQN	DUEL	PRIOR.	PRIOR. DUEL.
Alien	8.1%	14.5%	**21.8%**	19.3%	11.1%
Amidar	10.9%	10.3%	10.5%	7.7%	**14.8%**
Assault	719.2%	1275.9%	828.6%	1381.5%	**2334.4%**
Asterix	40.8%	226.2%	212.7%	302.8%	**4938.4%**
Asteroids	1.6%	0.9%	**3.3%**	2.5%	0.4%
Atlantis	2128.0%	2335.5%	**3293.9%**	2419.0%	3125.3%
Bank Heist	46.7%	138.8%	**177.8%**	137.3%	157.8%
Battle Zone	68.5%	71.9%	**94.2%**	74.5%	91.9%
Beam Rider	64.5%	116.7%	97.5%	210.3%	**252.7%**
Berzerk	14.6%	39.9%	35.0%	32.8%	**97.1%**
Bowling	19.2%	**30.9%**	27.5%	15.1%	13.7%
Boxing	648.2%	677.0%	711.2%	666.7%	**728.5%**
Breakout	1341.9%	1396.7%	**1559.0%**	1298.3%	1342.4%
Centipede	24.4%	23.0%	35.2%	18.6%	**43.4%**
Chopper Command	52.8%	34.4%	37.9%	48.2%	**89.5%**
Crazy Climber	380.6%	447.7%	493.9%	506.5%	**508.0%**
Defender	113.2%	207.2%	259.8%	176.0%	**263.2%**
Demon Attack	381.6%	2151.6%	1734.8%	1888.0%	**2261.9%**
Double Dunk	622.5%	982.5%	948.7%	**1998.7%**	328.7%
Enduro	86.2%	158.0%	262.7%	232.7%	**280.5%**
Fishing Derby	91.8%	97.7%	88.8%	105.7%	**114.5%**
Freeway	105.1%	112.5%	0.6%	**113.1%**	110.2%
Frostbite	10.4%	33.4%	54.8%	83.2%	**96.0%**
Gopher	385.3%	727.9%	960.8%	1679.2%	**5089.7%**
Gravitar	**1.8%**	-1.6%	1.8%	0.8%	-2.7%
H.E.R.O.	55.3%	54.9%	56.2%	**79.6%**	57.2%
Ice Hockey	79.2%	70.8%	82.4%	92.5%	**99.4%**
James Bond	198.2%	161.0%	239.4%	**1172.4%**	164.6%
Kangaroo	166.6%	420.8%	387.8%	**457.9%**	28.8%
Krull	528.0%	589.7%	**720.8%**	597.7%	679.8%
Kung-Fu Master	100.5%	146.0%	117.1%	153.2%	**181.5%**
Montezuma's Revenge	0.5%	0.4%	-0.1%	**0.6%**	-0.0%
Ms. Pac-Man	5.9%	6.9%	**13.5%**	11.0%	5.3%
Name This Game	98.9%	142.9%	186.9%	173.3%	**235.5%**
Phoenix	114.4%	202.3%	347.2%	284.0%	**1125.1%**
Pitfall!	3.7%	2.6%	**4.8%**	-1.2%	1.7%
Pong	107.6%	**110.9%**	110.0%	110.1%	108.6%
Private Eye	-0.7%	-1.9%	-0.6%	0.0%	**1.0%**
Q*Bert	76.4%	91.1%	**117.6%**	82.0%	116.6%
River Raid	30.2%	74.3%	**115.9%**	81.3%	115.3%
Road Runner	524.3%	643.2%	**873.7%**	779.6%	815.1%
Robotank	863.3%	868.7%	**913.3%**	824.4%	341.1%
Seaquest	10.0%	35.5%	**92.4%**	62.8%	3.0%
Skiing	27.1%	32.7%	29.0%	**44.1%**	-31.6%
Solaris	-8.4%	-13.8%	-3.1%	**2.5%**	-19.7%
Space Invaders	86.7%	190.8%	453.1%	290.8%	**685.9%**
Star Gunner	591.9%	653.0%	1020.3%	689.4%	**1431.0%**
Surround	24.7%	76.9%	91.1%	**103.2%**	62.9%
Tennis	**220.8%**	92.1%	175.0%	109.6%	55.6%
Time Pilot	63.7%	**140.3%**	140.0%	113.2%	67.2%
Tutankham	26.2%	63.3%	28.1%	35.2%	**76.4%**
Up and Down	79.8%	200.0%	**261.8%**	124.6%	239.1%
Venture	11.6%	0.3%	**17.8%**	7.4%	1.1%
Video Pinball	987.2%	2351.6%	709.5%	1892.3%	**2860.5%**
Wizard Of Wor	21.5%	143.8%	166.6%	131.2%	**257.6%**
Yars' Revenge	6.8%	10.5%	53.7%	7.0%	**124.1%**
Zaxxon	49.4%	101.9%	121.6%	112.9%	**136.1%**
Mean	219.6%	332.9%	343.8%	386.7%	**567.0%**
Median	68.5%	110.9%	**117.1%**	112.9%	115.3%

基于 Double Q-learning 的深度强化学习[1,2]

摘要

流行的 Q-learning 算法在特定条件下会过优估计动作价值。以往我们并不知道在实践中这种过优估计是否普遍，是否会损害性能，以及是否可以被普遍避免。在本文中，我们对所有这些问题都给出了肯定的回答。具体来说，我们首先证明了最近的 DQN 算法（它将 Q-learning 与深度神经网络相结合）在 Atari 2600 领域的一些游戏中存在严重的过优估计问题。然后，我们证明了在表格设置中引入的 Double Q-learning 算法的思想可以推广到大规模函数逼近中。我们提出了一种对 DQN 算法的特定改编，并证明了由此产生的算法不仅减少了观察到的过优估计，而且还在几个游戏中带来了更好的性能。

1 引言

强化学习的目标（Sutton and Barto 1998）是通过优化累积的未来奖励信号，为序列决策问题学习好的策略。Q-learning（Watkins 1989）是最流行的强化学习算法之一，但众所周知，它有时会学习到不切实际的高动作价值，因为它包含一个对估计动作价值函数的最大化步骤，因此倾向于选择被过优估计的价值而非被低估的价值。

在以往的研究中，过优估计被归因于不够灵活的函数逼近（Thrun and Schwartz 1993）和噪声（van Hasselt 2010, 2011）。在本文中，我们统一了这些观点，并指出当动作价值函数不准确时，无论近似误差的来源如何，都可能发生过优估计。当然，不精确的价值估计在学习过程中是常态，这表明过优估计可能比以前认识到的要普遍得多。

一个悬而未决的问题是，如果确实发生了过优估计，这是否会在实践中对性能产生负面影响。过于乐观的价值估计本身不一定是个问题。如果所有的价值都统一地偏高，那么行为的相对偏好将被保留，我们不会预期最终的策略会变得更差。此外，有时保持乐观是有益的：面对不确定性时的乐观主义是一种常见的探索技术（Kaelbling et al. 1996）。然而，如果过优估计不是统一的，也没有集中在我们

[1]原文：Van Hasselt Hado, Arthur Guez, and David Silver. ”Deep reinforcement learning with double Q-learning.” Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[2]译者：于梅灵。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

希望更多了解的状态上，那么它们可能会对最终策略的质量产生负面影响。Thrun and Schwartz (1993) 给出了一些具体例子，在这些例子中，即使在渐近情况下，这也会导致次优策略。

为了测试过优估计是否在实践中大规模发生，我们研究了最近的 DQN 算法（Mnih et al. 2015）的性能。DQN 将 Q-learning 与一个灵活的深度神经网络相结合，并在一个多样化且庞大的确定性 Atari 2600 游戏集上进行了测试，在许多游戏上达到了人类水平的性能。在某些方面，这种设置为 Q-learning 提供了一个最佳场景，因为深度神经网络提供了灵活的函数逼近，具有潜在的低渐近逼近误差，而且环境的确定性避免了噪声的有害影响。也许令人惊讶的是，我们发现即使在这种相对有利的设置下，DQN 有时也会大幅过优估计行为的价值。

我们证明了最初在表格设置中提出的 Double Q-learning 算法（van Hasselt 2010）可以推广到任意函数逼近，包括深度神经网络。我们用它来构建一个名为 Double DQN 的新算法。该算法不仅能产生更准确的价值估计，而且在几个游戏上取得了更高的分数。这表明 DQN 的过优估计确实会导致较差的策略，并且减少这种过优估计是有益的。此外，通过对 DQN 的改进，我们在 Atari 领域取得了最先进的结果。

2 背景

为了解决序列决策问题，我们可以学习每个动作的最优价值的估计值，该值定义为采取该动作并随后遵循最优策略时未来奖励的期望总和。在给定策略 π 下，状态 s 中动作 a 的真实价值是

$$Q_\pi(s,a) = \mathbb{E}[R_1 + \gamma R_2 + \ldots | S_0 = s, A_0 = a, \pi],$$

其中 $\gamma \in [0,1]$ 是一个折扣因子，用于权衡即时奖励和未来奖励的重要性。最优价值则为 $Q_*(s,a) = \max_\pi Q_\pi(s,a)$。通过在每个状态中选择价值最高的动作，可以很容易地从最优价值中推导出最优策略。

最优动作价值函数的估计可以通过使用 Q-learning（Watkins 1989）来学习，这是一种时间差分方法（Sutton 1988）。大多数有趣的问题都太大，无法独立学习所有状态下的所有动作价值函数。取而代之的是，我们可以学习一个参数化的价值函数 $Q(s,a;\theta_t)$。在状态 S_t 中采取动作 A_t 并观察到即时奖励 R_{t+1} 和下一时刻状态 S_{t+1} 后，参数化的标准 Q-learning 更新为

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t), \tag{1}$$

其中 α 是一个标量步长，目标 Y_t^Q 定义为

$$Y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t). \tag{2}$$

这个更新类似于随机梯度下降，将当前值 $Q(S_t, A_t; \theta_t)$ 朝着目标值 Y_t^Q 更新。

2.1 DQN

DQN 是一个多层神经网络，对于给定的状态 s，它输出一个动作价值向量 $Q(s, \cdot; \theta)$，其中 θ 是网络的参数。对于一个 n 维状态空间和一个包含 m 维的动作空间，神经网络是一个从 $\mathbb{R}^n$ 到 $\mathbb{R}^m$ 的函数。Mnih 等人（2015）提出的 DQN 算法的两个重要组成部分是目标网络和经验回放。

目标网络，其参数为 θ^-，与在线网络相同，只是其参数每 τ 步从在线网络复制一次，因此届时 $\theta_t^- = \theta_t$，并在所有其他步骤中保持固定。DQN 使用的目标则是

$$Y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-). \tag{3}$$

对于经验回放（Lin 1992），观察到的状态动作转移被存储一段时间，并从该记忆库中均匀采样以更新网络。目标网络和经验回放都极大地提高了算法的性能（Mnih et al. 2015）。

2.2 Double Q-learning

在 (2) 和 (3) 中，标准 Q-learning 和 DQN 中的最大化算子，使用相同的值来选择和评估一个行为。这使得它更有可能选择被过优估计的值，从而导致过于乐观的价值估计。为了防止这种情况，我们可以将选择与评估解耦。

在 Double Q-learning 中（van Hasselt 2010），通过将经验随机分配给两个价值函数之一进行更新来学习两个价值函数，从而产生两组权重，θ 和 θ'。对于每次更新，一组权重用于确定贪婪策略，另一组用于确定其价值。为了进行清晰的比较，我们可以解开 Q-learning 中的选择和评估，并将其目标 (2) 重写为

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t).$$

Double Q-learning 的目标则可以写成

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t'). \tag{4}$$

请注意，在 argmax 中，动作的选择仍然取决于在线权重 θ_t。这意味着，和在 Q-learning 中一样，我们仍然是根据由 θ_t 定义的当前值来估计贪婪策略的价值。但是，我们使用第二组权重 θ_t' 来公正地评估该策略的价值。这第二组权重可以通过交换 θ 和 θ' 的角色来对称地更新。

3 由估计误差引起的过度乐观

Q-learning 的过度估计问题最早由 Thrun 和 Schwartz（1993）研究，他们指出，如果动作价值包含在区间 $[-\epsilon, \epsilon]$ 内均匀分布的随机误差，那么每个目标的最高估值可达 $\gamma\epsilon\frac{m-1}{m+1}$，其中 m 是动作的数量。此外，Thrun 和 Schwartz 给出了一个具体的例子，其中这些过优估计即使在渐近情况下也会导致次优策略，并展示了在使用函数逼近时，过优估计在一个小型玩具问题中如何表现出来。Van Hasselt（2010）指出，即使使用表格表示，环境中的噪声也可能导致过优估计，并提出了 Double Q-learning 作为解决方案。在本节中，我们更普遍地证明，任何类型的估计误差都可能引起向上的偏差，无论这些误差是由于环境噪声、函数逼近、非平稳性还是任何其他来源导致的。这一点很重要，因为在实践中，任何方法在学习过程中都会产生一些不准确性，这仅仅是因为真实价值最初是未知的。上面引用的 Thrun 和 Schwartz（1993）的结果给出了特定设置下过度估计的上限，但也可以推导出过优估计的下限。

定理 1. *考虑一个特殊的状态 s，在该状态下，所有动作的真实最优价值都相等，我们将其记为 $V_*(s)$（即对所有动作 a 都有 $Q_*(s,a) = V_*(s)$）。假设在时间步 t 我们有一组动作价值的估计值 $Q_t(s,a)$，这组估计满足以下两个条件：*

1. *总体无偏: 估计误差的总和为零，即 $\sum_a (Q_t(s,a) - V_*(s)) = 0$。这意味着估计值在 $V_*(s)$ 周围波动，没有系统性的向上或向下偏移。*
2. *存在估计误差: 这组估计值并非完全准确，其均方误差为一个正常数 C，即 $\frac{1}{m}\sum_a (Q_t(s,a) - V_*(s))^2 = C$。这里 $m \geq 2$ 是状态 s 下动作的数量。*

在这些条件下，对于标准的 Q-learning，其单步更新中使用的最大价值估计值存在一个正的下界：

$$\max_a Q_t(s,a) \geq V_*(s) + \sqrt{\frac{C}{m-1}},$$

且这个下界是紧的。然而，在完全相同的条件下，Double Q-learning 估计的绝对误差下限为零。

证明： 将每个动作的误差定义为 $\epsilon_a = Q_t(s,a) - V_*(s)$。我们用反证法进行证明。假设存在一个 ϵ_a 的设定，使得 $\max_a \epsilon_a < \sqrt{\frac{C}{m-1}}$。设 $\{\epsilon_i^+\}$ 表示元素个数为 n 的非负 ϵ_a 集合，$\{\epsilon_j^-\}$ 表示元素个数为 $m-n$ 的严格负 ϵ_a 集合（因此 $\{\epsilon_a\} = \{\epsilon_i^+\} \cup \{\epsilon_j^-\}$）。若 $n = m$，则 $\sum \epsilon_a = 0 \implies \epsilon_a = 0, \forall a$，这与 $\sum \epsilon_a^2 = mC$ 的前提相矛盾。因此，必须有 $n \leq m-1$。此外，根据 $\max_a \epsilon_a < \sqrt{\frac{C}{m-1}}$ 可得 $\sum_{i=1}^n \epsilon_i^+ \leq n \max_i \epsilon_i^+ < n\sqrt{\frac{C}{m-1}}$，并且利用约束条件 $\sum \epsilon_a = 0$ 可以推断出 $|\sum_{j=1}^{m-n} \epsilon_j^-| < n\sqrt{\frac{C}{m-1}}$，则有 $\max_j |\epsilon_j^-| <$

$n\sqrt{\frac{C}{m-1}}$。因此可以得出：

$$\sum_{j=1}^{m-n}(\epsilon_j^-)^2 \leq \left|\sum_{j=1}^{m-n}\epsilon_j^-\right| \cdot \max_j|\epsilon_j^-| < n\sqrt{\frac{C}{m-1}}n\sqrt{\frac{C}{m-1}}.$$

下面可以结合这些关系来计算所有 ϵ_a 的平方和的上限：

$$\begin{aligned}\sum_{a=1}^{m}(\epsilon_a)^2 &= \sum_{i=1}^{n}(\epsilon_i^+)^2 + \sum_{j=1}^{m-n}(\epsilon_j^-)^2 \\ &< n\frac{C}{m-1} + n\sqrt{\frac{C}{m-1}}n\sqrt{\frac{C}{m-1}} \\ &= \frac{Cn(n+1)}{m-1} \\ &\leq mC.\end{aligned}$$

这与 $\sum_{a=1}^{m}\epsilon_a^2 = mC$ 的假设相矛盾，因此对于所有满足约束条件的 ϵ_a 设定，都有 $\max_a\epsilon_a \geq \sqrt{\frac{C}{m-1}}$。我们可以设定 $\epsilon_k = \sqrt{\frac{C}{m-1}}$ 对于 $k = 1,\ldots,m-1$ 和 $\epsilon_m = -\sqrt{(m-1)C}$，该设定满足 $\sum\epsilon_a^2 = mC$ 和 $\sum\epsilon_a = 0$ 这两个约束条件，且 $\max_a\epsilon_a = \sqrt{\frac{C}{m-1}}$，因此这个下限是紧的。对于 Double Q-learning，其绝对误差 $|Q'_t(s,\text{argmax}_a Q_t(s,a)) - V_*(s)|$ 的下限是零可以从下面的例子看出，设定：

$$Q_t(s,a_1) = V_*(s) + \sqrt{C(m-1)},$$

$$Q_t(s,a_i) = V_*(s) - \sqrt{\frac{C}{(m-1)}}, \quad \text{对于} i > 1.$$

这样定理的条件就成立了。虽然 Double Q-learning 算法在整个学习过程中，θ_t 和 θ'_t 不是相互独立的（它们从相同的经验中学习，并且其更新过程是相互依赖的），但是在每一次具体的更新步骤中，用于选择动作的 Q_t 网络和用于评估价值的 Q'_t 网络是解耦的，因此在单次更新中 Q_t 和 Q'_t 的估计误差是相互独立的，我们可以令 $Q'_t(s,a_1) = V_*(s)$，即 Q'_t 的估计误差为零，此时绝对误差也为零，而其余的动作价值 $Q'_t(s,a_i)$（对于 $i>1$）可以是任意值。 □

注 2. 定理1在一个理想化的设定下，为 Q-Learning 中的过高估计问题提供了一个明确的数学下限。该设定依赖于两个核心假设：1) 对于状态 s，所有动作 a 都是最优的（即 $Q_*(s,a) = V_*(s)$）；2) 对 $V_*(s)$ 的估计是无偏的（即 $\sum_a(Q_t(s,a) - V_*(s)) = 0$）。下面我们将探讨当放宽这两个理想化假设时，该定理的结论会发生何种变化。

考虑以下两个修改：1) 允许次优动作的存在：我们不再假设所有动作都是最优的。在状态 s 中，存在一个或多个最优动作满足 $Q_*(s,a^*) = V_*(s)$，以及一系列次优动作满足 $Q_*(s,a') < V_*(s)$。2) 引入系统性偏差：我们将无偏假设替换为一个

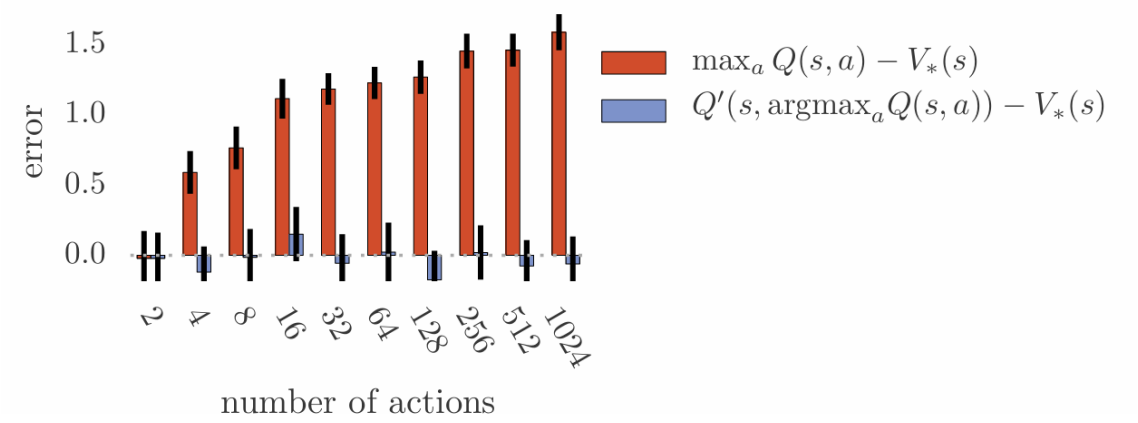


图 0.17　橙色条显示了当动作价值为 $Q(s,a) = V_*(s) + \epsilon_a$ 且误差 $\{\epsilon_a\}_{a=1}^m$ 为独立标准正态随机变量时，单次 Q-learning 更新中的偏差。用于蓝色条的第二组动作价值 Q' 是以相同且独立的方式生成的。所有条形图均为 100 次重复的平均值。

更广义的设定，允许 Q 值估计相对于真实最优状态值存在一个系统性的总偏差 δ，即 $\sum_a (Q_t(s,a) - V_*(s)) = \delta$。$\delta$ 可以为正、为负或为零。在新的设定下，我们关注的核心量 $\epsilon_a = Q_t(s,a) - V_*(s)$ 不再是纯粹的估计误差。它现在是一个复合量：

$$\epsilon_a = \underbrace{(Q_t(s,a) - Q_*(s,a))}_{\text{纯粹估计误差}} + \underbrace{(Q_*(s,a) - V_*(s))}_{\text{次优偏差（}\leq 0\text{）}}.$$

与此同时，集合 $\{\epsilon_a\}$ 满足总偏差 $\delta = \sum_a \epsilon_a$ 和均方误差 $C = \frac{1}{m}\sum_a \epsilon_a^2$ 两个约束。尽管问题的内在含义变得更加复杂，我们依然可以为 $\max_a \epsilon_a = \max_a (Q_t(s,a) - V_*(s))$ 推导出一个新的数学下限。在满足约束条件 $\sum_a \epsilon_a = \delta$ 和 $\sum_a \epsilon_a^2 = mC$ 的情况下，过高估计的下限由以下不等式给出：

$$\max_a Q_t(s,a) \geq V_*(s) + \frac{\delta}{m} + \frac{1}{m}\sqrt{\frac{m^2 C - \delta^2}{m-1}}.$$

新的下限包含一个 δ/m 项，它明确指出，系统性的正向偏差（$\delta > 0$）会直接抬高过高估计的基线，这符合直觉。

请注意，我们不需要假设不同动作的估计误差是独立的。该定理表明，即使价值估计平均是正确的，任何来源的估计误差都可能将估计值推高，使其偏离真实的最优值。定理1中的下限随着动作数量的增加而减小。这是考虑下限所导致的人为结果，因为它要求达到非常特定的值。更典型地，过度乐观随着动作数量的增加而增加，如图0.17所示。Q-learning 的过度估计确实随着动作数量的增加而增加，而 Double Q-learning 是无偏的。

考虑一个状态 s，在该状态下所有真实的最优动作价值均相等，即 $Q_*(s,a) = V_*(s)$。假设估计误差 $Q_t(s,a) - Q_*(s,a)$ 是在区间 $[-1,1]$ 上独立同分布的均匀随机

变量。那么，过优估计偏差的期望值为：

$$\mathbb{E}\left[\max_a Q_t(s,a) - V_*(s)\right] = \frac{m-1}{m+1}.$$

证明： 首先，我们将误差定义为 $\epsilon_a = Q_t(s,a) - Q_*(s,a)$。根据定理的假设，$\epsilon_a$ 是一个在 $[-1,1]$ 上均匀分布的随机变量。我们的目标是求解 $\mathbb{E}[\max_a \epsilon_a]$。为此，我们首先推导随机变量 $\max_a \epsilon_a$ 的累积分布函数（CDF）。事件 $\max_a \epsilon_a \le x$ 发生的概率，等价于所有独立的误差项 ϵ_a 同时满足 $\epsilon_a \le x$ 的概率。由于各项误差是相互独立的，则有：

$$P(\max_a \epsilon_a \le x) = P(\epsilon_1 \le x \wedge \epsilon_2 \le x \wedge \cdots \wedge \epsilon_m \le x) = \prod_{a=1}^{m} P(\epsilon_a \le x),$$

其中，$P(\epsilon_a \le x)$ 是单个误差项 ϵ_a 的 CDF。对于 $[-1,1]$ 上的均匀分布，其 CDF 定义为：

$$P(\epsilon_a \le x) = \begin{cases} 0, & 若x < -1. \\ \frac{1+x}{2}, & 若x \in [-1,1). \\ 1, & 若x \ge 1. \end{cases}$$

因此，我们可以得到最大值 $\max_a \epsilon_a$ 的 CDF 为：

$$P(\max_a \epsilon_a \le x) = \prod_{a=1}^{m} P(\epsilon_a \le x) = \begin{cases} 0, & 若x < -1. \\ \left(\frac{1+x}{2}\right)^m, & 若x \in [-1,1). \\ 1, & 若x \ge 1. \end{cases}$$

这样，我们就得到了随机变量 $\max_a \epsilon_a$ 的 CDF。根据期望的定义，其期望值可以通过以下积分计算：

$$\mathbb{E}\left[\max_a \epsilon_a\right] = \int_{-1}^{1} x f_{\max}(x) dx,$$

其中 $f_{\max}(x)$ 是该变量的概率密度函数（PDF），由其 CDF 求导得出。对于 $x \in [-1,1]$，我们有：

$$f_{\max}(x) = \frac{d}{dx} P(\max_a \epsilon_a \le x) = \frac{m}{2}\left(\frac{1+x}{2}\right)^{m-1}.$$

计算该定积分可得：

$$\begin{aligned} \mathbb{E}\left[\max_a \epsilon_a\right] &= \int_{-1}^{1} x f_{\max}(x) dx \\ &= \left[\left(\frac{x+1}{2}\right)^m \frac{mx-1}{m+1}\right]_{-1}^{1} \end{aligned}$$

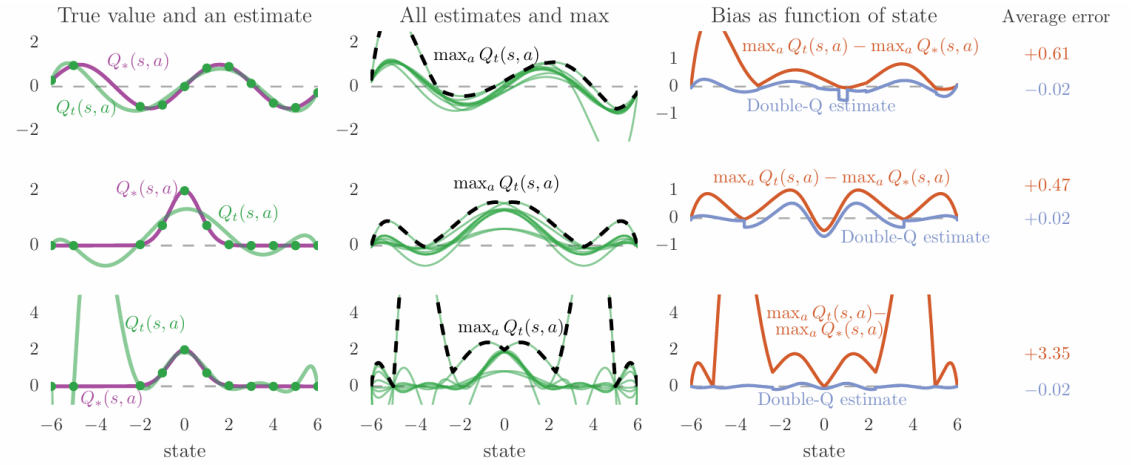


图 0.18　学习过程中过优估计的图示。在每个状态（x 轴）中，有 10 个动作。**左列**显示了真实价值 $V_*(s)$（紫线）。所有真实动作价值都由 $Q_*(s,a) = V_*(s)$ 定义。绿线显示了一个动作的估计值 $Q(s,a)$ 作为状态的函数，该值是在几个采样状态（绿点）上对真实值进行拟合得到的。**中间列**的图显示了所有的估计值（绿色），以及这些值的最大值（黑色虚线）。最大值几乎在任何地方都高于真实值（紫色，左图）。**右列**的图用橙色显示了差异。右图中的蓝线是 Double Q-learning 在每个状态使用第二组样本所得到的估计值。蓝线更接近于零，表明偏差较小。三**行**对应于不同的真实函数（左，紫色）或拟合函数的容量（左，绿色）。（详情见正文）

$$
\begin{aligned}
&= \left(\left(\frac{1+1}{2}\right)^m \frac{m(1)-1}{m+1}\right) - \left(\left(\frac{-1+1}{2}\right)^m \frac{m(-1)-1}{m+1}\right) \\
&= \left(1^m \cdot \frac{m-1}{m+1}\right) - (0) \\
&= \frac{m-1}{m+1}.
\end{aligned}
$$

□

我们现在转向函数逼近，并考虑一个实值连续状态空间，每个状态有 10 个离散动作。为简单起见，本例中的真实最优动作价值仅取决于状态，因此在每个状态下所有动作都具有相同的真实价值。这些真实值显示在图0.18的左列图中（紫色线），定义为 $Q_*(s,a) = \sin(s)$（顶行）或 $Q_*(s,a) = 2\exp(-s^2)$（中行和底行）。左图还显示了单个动作的近似值（绿线）作为状态的函数，以及估计所依据的样本（绿点）。该估计是一个 d 次多项式，它拟合于采样状态下的真实值，其中 d=6（顶行和中行）或 d=9（底行）。样本与真实函数完全匹配：没有噪声且我们假设在这些采样状态下拥有动作价值的真实情况。对于前两行，即使在采样状态上，近似也是不精确的，因为函数逼近的灵活性不足。在底行，函数足够灵活以拟合绿点，但这降低了在未采样状态下的准确性。请注意，在左图的左侧附近，采样状态的间距更大，导致更大的估计误差。在许多方面，这是一个典型的学习设置，在每个时间点我们都只有有限的数据。

图0.18的中间列图表显示了所有 10 个动作的估计动作值（绿线），作为状态的

函数，以及每个状态下的最大动作值（黑色虚线）。虽然所有动作的真实价值函数是相同的，但由于它们基于不同的采样状态集，所以近似值有所不同。最大值通常高于左侧紫色显示的真实值。这在右图中得到了证实，该图用橙色显示了黑色和紫色曲线之间的差异。橙色线几乎总是正的，表明存在向上的偏差。右图还用蓝色显示了 Double Q-learning 的估计值，这些估计值平均更接近于零。这表明 Double Q-learning 确实可以成功地减少 Q-learning 的过优估计。

图0.18中的不同行显示了同一实验的变体。顶行和中行之间的区别在于真实价值函数，这表明过度估计并非特定真实价值函数的产物。中行和底行之间的区别在于函数逼近的灵活性。在左中图中，由于函数灵活性不足，即使对于某些采样状态，估计也是不正确的。在左下图中的函数更灵活，但这对于未见过的状态会产生更高的估计误差，从而导致更高的过优估计。这一点很重要，因为灵活的参数化函数逼近常用于强化学习（例如，Tesauro 1995, Sallans and Hinton 2004, Riedmiller 2005, and Mnih et al. 2015）。

与 Van Hasselt（2010）不同，我们没有使用统计论证来发现过优估计，获得图0.18的过程是完全确定性的。与 Thrun 和 Schwartz（1993）不同，我们不依赖于具有不可约渐近误差的不灵活函数逼近；底行显示，一个足够灵活以覆盖所有样本的函数会导致高度的过优估计，这表明过度估计可以很普遍地发生。

在上面的例子中，即使假设我们在某些状态下拥有真实动作价值的样本，过优估计也会发生。如果我们从已经过度乐观的动作价值进行引导（bootstrap），价值估计可能会进一步恶化，因为这会导致过优估计在我们的估计中传播。虽然统一过优估计价值可能不会损害最终的策略，但在实践中，不同状态和动作的过优估计误差会有所不同。过优估计与引导相结合会产生有害影响，即传播关于哪些状态比其他状态更有价值的错误相对信息，从而直接影响学习策略的质量。

这种过优估计不应与面对不确定性时的乐观主义相混淆（Sutton 1990, Agrawal 1995, Kaelbling et al. 1996, Auer et al. 2002, Brafman and Tennenholtz 2003, Szita and Lőrincz 2008, Strehl and Littman 2009），在那种情况下，会给价值不确定的状态或动作一个探索奖励。这里讨论的过优估计仅在更新后发生，导致在表面确定性面前的过度乐观。Thrun and Schwartz（1993）指出，与面对不确定性时的乐观主义相反，这些过优估计实际上会阻碍学习最优策略。我们在实验中证实了这种对策略质量的负面影响：当我们使用 Double Q-learning 减少过优估计时，策略得到了改善。

4 Double DQN

Double Q-learning 的思想是通过将目标中的最大化操作分解为动作选择和动作价值评估两部分来减少过优估计。虽然没有完全解耦，但 DQN 架构中的目标网络为第二个价值函数提供了一个自然的选择，而无需引入额外的网络。因此，我们建议根据在线网络来评估贪婪策略，但使用目标网络来估计其价值。结合 Double Q-learning 和 DQN，我们将由此产生的算法称为 Double DQN。它的更新与 DQN 相同，但将目标 Y_t^{DQN} 替换为

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t), \theta_t^-)$$

与 Double Q-learning (4) 相比，第二个网络 θ_t' 的权重被目标网络 θ_t^- 的权重所取代，用于评估当前的贪婪策略。对目标网络的更新与 DQN 一致，仍然是在线网络的周期性副本。这个版本的 Double DQN 可能是对 DQN 朝向 Double Q-learning 的最小可能改动。其目标是获得 Double Q-learning 的大部分好处，同时保持 DQN 算法的其余部分不变以便进行公平比较，并且计算开销最小。

5 实验结果

在本节中，我们分析 DQN 的过优估计，并证明 Double DQN 在价值准确性和策略质量方面均优于 DQN。为了进一步测试该方法的鲁棒性，我们还使用了由人类专家轨迹生成的随机起点来评估算法，正如 Nair 等人（2015）所建议的那样。我们的测试平台由 Atari 2600 游戏组成，使用了 Arcade 学习环境（Bellemare et al. 2013）。目标是让单个算法使用一组固定的超参数，仅通过屏幕像素作为输入，从交互中独立学习玩每个游戏。这是一个要求很高的测试平台：不仅输入是高维的，而且游戏视觉效果和游戏机制在不同游戏之间差异很大。因此，好的解决方案必须在很大程度上依赖于学习算法——仅仅依靠调优来对领域进行过拟合在实践中是不可行的。我们严格遵循 Mnih 等人（2015）使用的实验设置和网络架构。简而言之，网络架构是一个卷积神经网络（Fukushima 1988; Lecun et al. 1998），具有 3 个卷积层和一个全连接隐藏层（总共约 150 万个参数）。该网络将最后四帧作为输入，并输出每个动作的动作价值。在每个游戏上，网络在单个 GPU 上训练 2 亿帧。

5.1 关于过度乐观的结果

图0.19显示了 DQN 在六个 Atari 游戏中的过优估计示例。DQN 和 Double DQN 都是在 Mnih 等人（2015）描述的完全相同的条件下进行训练的。DQN 对当前贪婪策略的价值始终存在并且有时是巨大的过度乐观，通过将顶部图中的橙色学习曲线与代表最佳学习策略的实际折扣价值的橙色直线进行比较可以看出这一点。更

准确地说，（平均）价值估计是在训练期间定期计算的，完整的评估阶段长度为 $T = 125,000$ 步，计算公式为

$$\frac{1}{T}\sum_{t=1}^{T}\max_{a} Q(S_t, a; \theta).$$

基准真实的平均值是通过运行最佳学习策略数个回合并计算实际累积奖励获得的。如果没有过优估计，我们期望这些量能够匹配（即，曲线与每个图的直线相匹配）。然而，DQN 的学习曲线最终总是远高于真实值。Double DQN 的学习曲线（以蓝色显示）更接近代表最终策略真实价值的蓝色直线。请注意，蓝色直线通常高于橙色直线。这表明 Double DQN 不仅产生更准确的价值估计，而且还产生更好的策略。在中间两个图中显示了更极端的过优估计，其中 DQN 在 Asterix 和 Wizard of Wor 游戏上表现出高度不稳定性。请注意 y 轴上价值的对数刻度。底部两个图显示了这两个游戏对应的分数。请注意，中间图中 DQN 价值估计的增加与底部图中分数的下降相吻合。这再次表明，过优估计正在损害最终策略的质量。如果独立地看，人们可能会认为观察到的不稳定性与函数逼近下离线策略学习的内在不稳定性问题有关（Baird 1995, Tsitsiklis and Van Roy 1997, Maei 2011, Sutton et al. 2015）。然而，我们看到使用 Double DQN 的学习要稳定得多，这表明这些不稳定性的原因实际上是 Q-learning 的过度乐观。图0.19仅显示了几个例子，但在所有 49 个测试的 Atari 游戏中都观察到了 DQN 程度不同的过优估计。

5.2 学习策略的质量

过度乐观并不总会对学习到的策略质量产生负面影响。例如，DQN 在 Pong 游戏中尽管略微过优估计了策略价值，但仍实现了最优行为。然而，减少过优估计可以显著有益于学习的稳定性；我们在图0.19中看到了这方面的清晰例子。我们现在通过在 DQN 测试过的所有 49 个游戏上进行评估，来更普遍地评估 Double DQN 在策略质量方面有多大帮助。正如 Mnih 等人（2015）所述，每个评估回合开始时都会执行一个特殊的不操作（no-op）动作，该动作最多执行 30 次且不影响环境，从而为智能体提供不同的起点。评估期间的一些探索提供了额外的随机性。对于 Double DQN，我们使用了与 DQN 完全相同的超参数，以便进行一个仅专注于减少过优估计的对照实验。学习到的策略使用 ϵ-贪婪策略（其中 $\epsilon = 0.05$）进行评估，评估时间为 5 分钟的模拟器时间（18,000 帧）。分数是 100 个回合的平均值。Double DQN 和 DQN 之间唯一的区别在于目标，使用的是 $Y^{\text{DoubleDQN}}$ 而不是 Y^{DQN}。这种评估有些对抗性，因为所用的超参数是为 DQN 调优的，而不是为 Double DQN。为

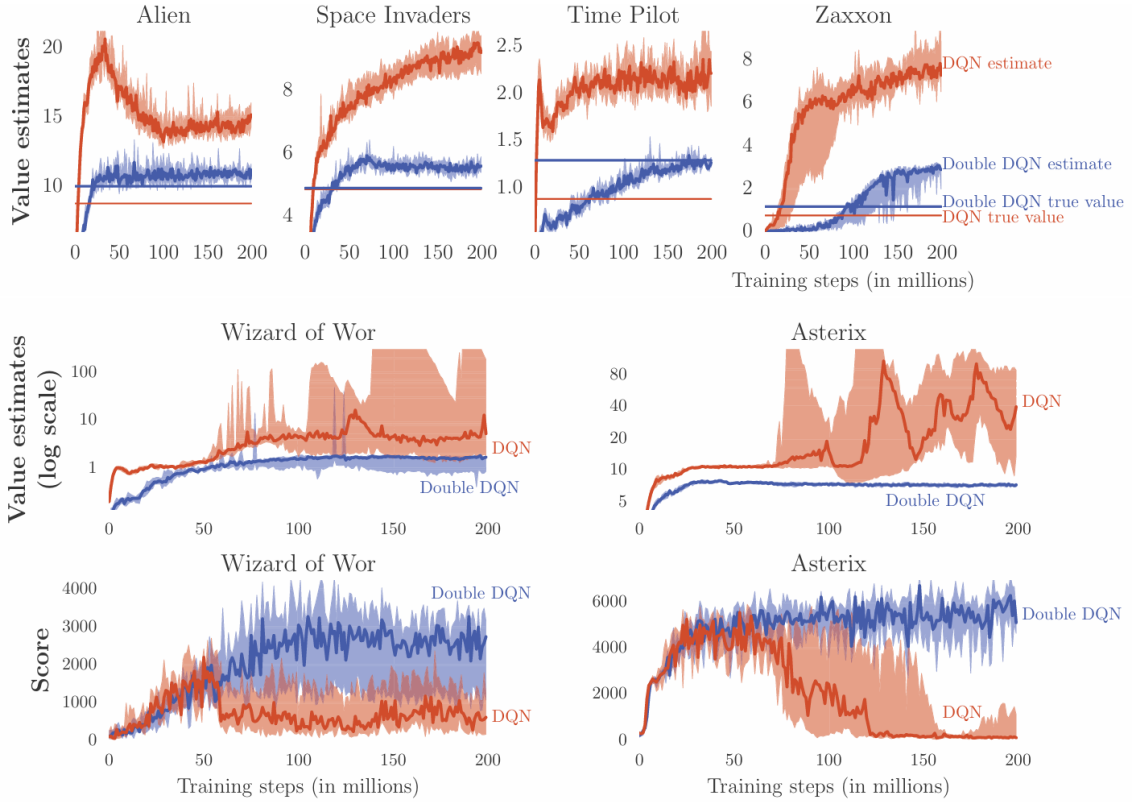


图 0.19　**顶行**和**中行**显示了 DQN（橙色）和 Double DQN（蓝色）在六个 Atari 游戏上的价值估计。该结果是通过使用 Mnih 等人（2015）采用的超参数，以 6 个不同的随机种子运行 DQN 和 Double DQN 获得的。较深的线显示了种子结果的中位数，我们对两个极值进行平均以获得阴影区域（即，使用线性插值的 10% 和 90% 分位数）。顶行中的水平橙色直线（代表 DQN）和蓝色直线（代表 Double DQN）是在学习结束后，通过运行相应的智能体，并对从每个访问过的状态获得的实际折扣回报进行平均计算得出的。如果没有偏差，这些直线将与图右侧的学习曲线相匹配。**中行**显示了两个 DQN 过度乐观情况非常极端的游戏的价值估计（使用对数刻度）。**底行**显示了在训练期间评估智能体时，这种过度乐观对所得分数的有害影响；当过优估计开始时，分数会下降，使用 Double DQN 的学习过程要稳定得多。

了获得跨游戏的汇总统计数据，我们按如下方式对每个游戏的分数进行归一化：

$$\text{score}_{\text{normalized}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{\text{score}_{\text{human}} - \text{score}_{\text{random}}} \tag{5}$$

“随机”和“人类”的分数与 Mnih 等人（2015）使用的相同，并在附录中给出。表 1 显示，在“no ops”条件下，Double DQN 整体上明显优于 DQN。详细的比较（见附录）表明，在几个游戏中，Double DQN 比 DQN 有很大的改进。值得注意的例子包括《Road Runner》（从 233% 提高到 617%）、《Asterix》（从 70% 提高到 180%）、《Zaxxon》（从 54% 提高到 111%）和《Double Dunk》（从 17% 提高到 397%）。Gorila 算法（Nair et al. 2015）是 DQN 的一个大规模分布式版本，由于其架构和基础设施的差异足以使直接比较变得不明确，因此未包含在表格中。为了完整起见，我们注意到 Gorila 获得的中位数和平均归一化分数分别为 96% 和 495%。

	no ops		human starts		
	DQN	DDQN	DQN	DDQN	DDQN (tuned)
Median	93%	**115%**	47%	88%	**117%**
Mean	241%	**330%**	122%	273%	**475%**

表 9　49 个游戏的归一化性能总结。测试包含两种条件：一是在每个回合开始时最多进行 30 次“无操作”（no ops）指令，持续长达 5 分钟；二是从随机选择的人类起点开始，持续长达 30 分钟。DQN 的结果分别来自 Mnih 等人（2015）的“无操作”测试和 Nair 等人（2015）的“人类起点”测试.

5.3 对人类起点的鲁棒性

先前评估的一个担忧是，在具有唯一起点的确定性游戏中，学习者可能学会在没有太多泛化需求的情况下记住动作序列。虽然成功，但该解决方案不会特别鲁棒。通过从不同的起点测试智能体，我们可以检验找到的解决方案是否具有良好的泛化能力，从而为学习到的策略提供一个具有挑战性的测试平台（Nair et al. 2015）。我们按照 Nair 等人（2015）的建议，从人类专家的轨迹中为每个游戏采样了 100 个起点。我们从每一个起点开始一个评估回合，并运行模拟器长达 108,000 帧（在 60Hz 下为 30 分钟，包括起点之前的轨迹）。每个智能体仅根据起点之后累积的奖励进行评估。对于本次评估，我们包含了一个调优版的 Double DQN。进行一些调优是合适的，因为超参数是为 DQN（一个不同的算法）调优的。对于调优版的 Double DQN，我们将目标网络的两次复制之间的帧数从 10,000 增加到 30,000，以进一步减少过优估计，因为在每次切换后，DQN 和 Double DQN 都会立即恢复到 Q-learning。此外，我们将学习期间的探索率从 $\epsilon = 0.1$ 降低到 $\epsilon = 0.01$，然后在评估期间使用 $\epsilon = 0.001$。最后，调优版本在网络的顶层为所有行为值使用一个单一的共享偏置。这些变化中的每一个都提高了性能，并且它们共同带来了明显更好的结果。表 1 报告了在 Mnih 等人（2015）的 49 个游戏上进行此评估（在人类起点下）的汇总统计数据。Double DQN 获得了明显更高的中位数和平均分数。同样，Gorila DQN（Nair et al. 2015）未包含在表格中，但为完整起见，请注意它获得的中位数为 78%，平均值为 259%。详细结果，以及另外 8 个游戏的结果，可在图0.20中找到。在一些游戏上，从 DQN 到 Double DQN 的改进是惊人的，在某些情况下，分数更接近人类水平，甚至超过了人类。Double DQN 在这种更具挑战性的评估中显得更鲁棒，这表明发生了适当的泛化，并且找到的解决方案没有利用环境的确定性。这很有吸引力，因为它表明在寻找通用解决方案方面取得了进展，而不是找到一个鲁棒性较差的确定性步骤序列。

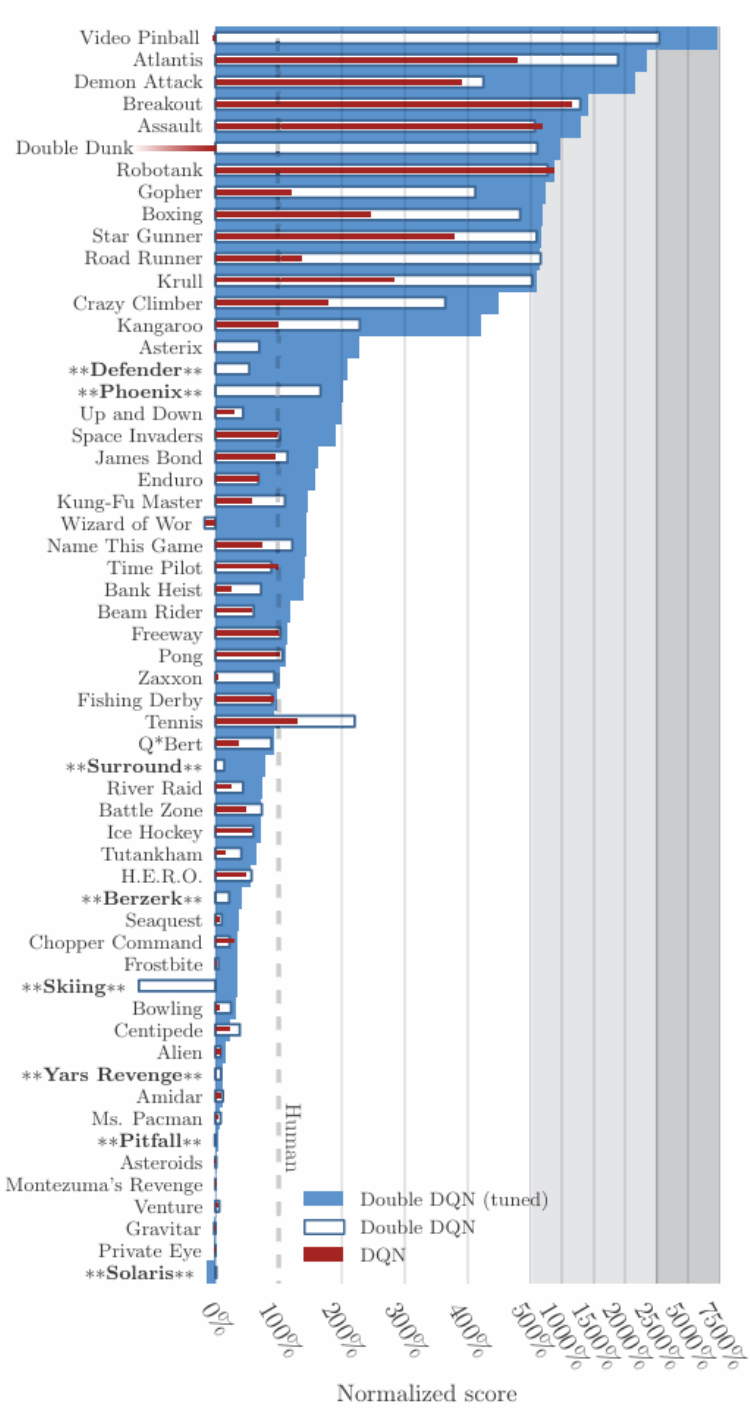


图 0.20　57 个 Atari 游戏的归一化分数。在人类起点条件下，每个游戏测试 100 个回合。与 Mnih 等人（2015）的研究相比，额外测试了八个游戏。这些新增的游戏用星号和粗体字体标出。

6 讨论

本文有五项贡献。第一，我们展示了为什么在大规模问题中，即使是确定性的问题，由于学习固有的估计误差，Q-learning 也可能出现过度乐观。第二，通过分析 Atari 游戏上的价值估计，我们表明这些过优估计在实践中比以前认识到的更普遍和严重。第三，我们证明了 Double Q-learning 可以大规模地用于成功减少这种过度乐观，从而实现更稳定和可靠的学习。第四，我们提出了一种名为 Double DQN 的具体实现，它利用了 DQN 算法的现有架构和深度神经网络，而无需额外的网络或参数。最后，我们证明了 Double DQN 能找到更好的策略，在 Atari 2600 领域取得了新的最先进成果。

致谢

我们要感谢 Tom Schaul、Volodymyr Mnih、Marc Bellemare、Thomas Degris、Georg Ostrovski 和 Richard Sutton 的有益评论，以及 Google DeepMind 的每一个人所营造的建设性研究环境。

参考文献

[1] R. Agrawal. SamplemeanbasedindexpolicieswithO(logn)regret for the multi-armed bandit problem. Advances in Applied Proba bility, pages 1054–1078, 1995.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. Machine learning, 47(2-3):235–256, 2002.

[3] L. Baird. Residual algorithms: Reinforcement learning with func tion approximation. In Machine Learning: Proceedings of the Twelfth International Conference, pages 30–37, 1995.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. J. Artif. Intell. Res. (JAIR), 47:253–279, 2013.

[5] R. I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. The Jour nal of Machine Learning Research, 3:213–231, 2003.

[6] K. Fukushima. Neocognitron: A hierarchical neural network capa ble of visual pattern recognition. Neural networks, 1(2):119–130, 1988.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. Journal of Artificial Intelligence Research,4: 237–285, 1996.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[9] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning, 8(3):293–321, 1992.

[10] H. R. Maei. Gradient temporal-difference learning algorithms. PhD thesis, University of Alberta, 2011.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostro vski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540): 529–533, 2015.

[12] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver. Massively par allel methods for deep reinforcement learning. In Deep Learning Workshop, ICML, 2015.

[13] M. Riedmiller. Neural fitted Q iteration- first experiences with a data efficient neural reinforcement learning method. In Proceed ings of the 16th European Conference on Machine Learning, pages 317–328. Springer, 2005.

[14] B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. The Journal of Machine Learning Research,5: 1063−1088, 2004.

[15] A. L. Strehl, L. Li, and M. L. Littman. Reinforcement learning in finite MDPs: PAC analysis. The Journal of Machine Learning Research, 10:2413−2444, 2009.

[16] R. S. Sutton. Learning to predict by the methods of temporal dif ferences. Machine learning, 3(1):9−44, 1988.

[17] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Pro ceedings of the seventh international conference on machine learn ing, pages 216−224, 1990.

[18] R. S. Sutton and A. G. Barto. Introduction to reinforcement learn ing. MIT Press, 1998.

[19] R. S. Sutton, A. R. Mahmood, and M. White. An emphatic ap proach to the problem of off-policy temporal-difference learning. arXiv preprint arXiv:1503.04269, 2015.

[20] I. Szita and A. L″orincz. The many faces of optimism: a unifying approach. In Proceedings of the 25th international conference on Machine learning, pages 1048−1055. ACM, 2008.

[21] G. Tesauro. Temporal difference learning and td-gammon. Com munications of the ACM, 38(3):58−68, 1995.

[22] S. Thrun and A. Schwartz. Issues in using function approxi mation for reinforcement learning. In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, Proceedings of the 1993 Connectionist Models Summer School, Hillsdale, NJ, 1993. Lawrence Erlbaum.

[23] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. IEEE Transactions on Au tomatic Control, 42(5):674−690, 1997.

[24] H. van Hasselt. Double Q-learning. Advances in Neural Informa tion Processing Systems, 23:2613−2621, 2010.

[25] H. van Hasselt. Insights in Reinforcement Learning. PhD thesis, Utrecht University, 2011.

[26] C. J. C. H. Watkins. Learning from delayed rewards. PhD thesis, University of Cambridge England, 1989.

附录

网络架构

实验中使用的卷积网络与 Mnih 等人 (2015) 提出的完全相同，此处提供其细节是为了内容的完整性。网络的输入是一个 $84x84x4$ 的张量，其中包含了最近四帧画面经过尺寸缩放和灰度化处理后的版本。第一个卷积层使用 32 个大小为 8（步

长为 4）的滤波器对输入进行卷积；第二个卷积层有 64 个大小为 4（步长为 2）的滤波器；最后一个卷积层有 64 个大小为 3（步长为 1）的滤波器。之后是一个拥有 512 个单元的全连接隐藏层。所有这些层之间都使用修正线性单元（ReLU）进行分隔。最后，一个全连接线性层将结果投影为网络的输出，即 Q 值。用于训练该网络的优化算法是 RMSProp（动量参数为 0.95）。

超参数

在所有实验中，折扣因子 γ 设为 0.99，学习率 α 设为 0.00025。目标网络两次更新之间的步数为 $\tau = 10,000$。训练总共进行 5000 万步（即 2 亿帧画面）。智能体每 100 万步评估一次，在这些评估中表现最好的策略将被保留下来，作为学习过程的最终输出。经验回放记忆库的大小为 100 万个元组。系统每 4 步从记忆库中采样一次，使用大小为 32 的小批量数据来更新网络。所使用的简单探索策略是 ϵ-贪婪策略，其参数 ϵ 在最初的 100 万步内从 1 线性下降到 0.1。

优先级经验回放 [1,2]

摘要

经验回放技术能让在线强化学习智能体能够记忆并复用过去的经验。在先前的研究中，经验转移（experience transitions）是从回放记忆池中均匀采样得到的。然而，这种方法只是按照经验最初发生时的频率来回放转移过程，完全不考虑其重要性。本文提出了一种经验优先级排序框架：通过更频繁地回放重要的转移过程，从而提升学习效率。我们将这种优先级经验回放应用于深度 Q 网络（DQN）——该算法曾在多款雅达利游戏中达到人类水平。结果表明，采用优先级经验回放的 DQN 创下新纪录：在 49 款游戏中，有 41 款游戏的表现超过了采用均匀回放的 DQN。

1 引言

在线强化学习（RL）智能体在观测经验流的过程中，会逐步更新其（策略、价值函数或模型的）参数。在最简单的实现形式中，智能体仅进行一次参数更新，就会立即丢弃新接收的数据。这种方式存在两个问题：（a）强相关性更新会破坏许多主流随机梯度类算法所依赖的独立同分布（i.i.d.）假设；（b）会快速遗忘那些可能在后续学习中发挥作用的稀有经验。

经验回放[1] 恰好解决了上述两个问题：将经验存储在回放记忆池中后，通过混合近期与远期经验进行参数更新，能够打破经验间的时间相关性；同时，稀有经验也不再仅用于单次更新，而是可以被多次利用。这一点在深度 Q 网络（DQN）算法[2,3] 中得到了验证——该算法通过使用经验回放，稳定了由深度神经网络表示价值函数的训练过程。具体来说，DQN 采用了一个具有大型滑动窗的回放记忆池，从中进行随机均匀采样，且每个经验转移平均会被重新利用 8 次（8 次是因为选择的小批量数为 32，回放周期为 4（见第 4 节）；能够节省资源消耗，增加稀有经验在训练中被使用的机会，同时不会过拟合）[3] 。总体而言，经验回放能够减少学习过程所需的经验总量，转而以更多的计算资源和存储资源作为替代——而相较于

[1]原文：Tom Schaul, John Quan, Ioannis Antonoglou and David Silver. Prioritized experience replay. In International Conference on Learning Representations, 2016. URL https://arxiv.org/abs/1511.05952.

[2]译者：谢君。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

[3]转移是强化学习中交互的基本单元，在我们的案例中，它是一个由（状态 S_{t-1}、动作 A_{t-1}、奖励 R_t、折扣 γ_t、后继状态 S_t）组成的元组。我们为简单起见选择了这种形式，但本文中的大多数论点，对于更粗粒度的经验分块方式（例如按序列或幕分块）也同样成立。

强化学习智能体与环境的交互成本，这些资源往往更为廉价。

在本文中，我们研究了对要回放的转移进行优先级排序，如何能让经验回放比均匀回放所有转移时更高效、更有效。核心思想是，RL 智能体从某些转移中学习的效果，可能比从其他转移中更好。转移可能或多或少令人惊讶、冗余，或者与任务相关。有些转移可能不会立即对智能体有用，但随着智能体能力的提升，可能会变得有用[4]。经验回放将在线学习智能体从必须按照经验发生的精确顺序处理转移的限制中解放出来。优先级经验回放进一步使智能体摆脱了以经验被获取时的频率去考虑转移样本的限制。

具体而言，我们建议更频繁地回放那些具有高预期学习进展的转移，这里用它们的时序差分（TD）误差（公式 (2)，预测的不准确程度）的大小来衡量。如果使用贪婪的优先级排序可能会导致多样性的损失（3.2 节，回放可能会集中于 TD 较大的经验，TD 误差较小的经验不易被回放，探索不全面），我们通过随机优先级排序来缓解这一问题（3.3 节）；同时还会引入偏差，我们通过重要性采样来纠正这种偏差（3.4 节）。我们得到的算法是鲁棒且可扩展的，这一点在雅达利 2600（Atari 2600）基准测试套件上得到了验证，在该套件上我们实现了更快的学习速度和最先进的性能。

2. 背景

大量神经科学研究已在啮齿动物的海马体中发现了经验回放的证据，表明先前经验的序列会在清醒休息或睡眠期间被回放。与奖励相关的序列似乎被更频繁地回放[5–7]。具有高幅度 TD 误差的经验也似乎被更频繁地回放[8,9]。

众所周知，像价值迭代这样的规划算法，通过以适当的顺序对更新进行优先级排序，可以变得更高效。优先清扫[10,11] 会根据执行更新产生的价值变化，来选择接下来要更新的状态。TD 误差提供了一种衡量这些优先级的方法[12]。我们的方法使用了一种类似的优先级排序方法，但针对的是无模型 RL，而非基于模型的规划。此外，当从样本中学习函数逼近器时，我们使用了一种随机优先级排序，这种排序更为鲁棒。

TD 误差也被用作一种优先级机制，以确定资源应聚焦于何处，例如在选择探索地点[13] 或选择哪些特征[14,15] 时。

在监督学习中，当类别标识已知时，有大量技术可用于处理不平衡数据集，包括重采样、欠采样和过采样技术，这些技术可能与集成方法相结合（有关综述请参见[16]）。最近有一篇论文在带有经验回放的深度强化学习背景下引入了一种重采样形式[17]；该方法将经验分为两个桶，一个用于正奖励，一个用于负奖励，然后从

每个桶中选取固定比例的经验。这仅适用于（与我们的情况不同的）存在“正/负”经验自然概念的领域。此外，[18] 引入了一种基于误差的非均匀采样形式，并带有重要性采样校正，这使得 MNIST 数字分类的速度提高了 3 倍。

目前已有几种利用深度强化学习玩雅达利游戏的方法被提出，包括 DQN[2,3,19–22]，以及 Double DQN 算法[23]，后者是当前已发表的最先进方法。与我们的工作的同期，一种将优势与价值函数分离的架构创新[24] 也在雅达利基准测试中带来了显著改进。

3 优先级回放

使用回放池会涉及两个层面的设计选择：存储哪些经验，以及回放哪些经验（以及如何回放）。本文仅针对后者：假设回放记忆池的内容不受我们控制，如何最有效地利用回放记忆池进行学习（也可参见第 6 节）。

3.1 一个启发性示例

均匀采样带来的问题：为了理解优先级排序的潜在收益，我们引入一个人工的“盲走悬崖（Blind Cliffwalk）”环境（如图 1 左侧所示），该环境例证了当稀疏奖励时探索所面临的挑战。仅有 n 个状态的情况下，该环境需要指数级数量的随机步骤才能获得第一个非零奖励；准确地说，一个随机动作序列导致奖励的概率是 2^{-n}。此外，最相关的转移（来自罕见的成功案例）隐藏在大量高度冗余的失败案例中（类似于双足机器人在学会行走之前反复摔倒）。(若采用随机策略从初始状态开始走 n 步，能获得有用的信息（奖励非 0）的可能性为 2^{-n}，也就是说，假如我们把各种转移都储存起来，然后采用均匀采样来取转移，仅有 2^{-n} 的概率取到有用的信息，这样的话学习效率就会很低。)

实验对比：我们用这个例子来突出两个智能体学习时间的差异。两个智能体都对从同一回放记忆池中抽取的转移进行 Q-学习更新。第一个智能体以随机均匀的方式回放转移，而第二个智能体调用一个“oracle”来对转移进行优先级排序。这个“oracle”会贪婪地选择在当前状态下能最大程度降低全局损失的转移（事后看来，是在参数更新之后）。有关设置的详细信息，请参见附录 B.1。图 1（右侧）显示，以良好的顺序选择转移，相较于均匀选择的方式，能带来指数级的速度提升。当然，这样的“oracle”（理想的优先级选择机制）在现实中并不存在，但这种巨大的性能差距促使我们去寻找一种能改进均匀随机回放的实用方法。

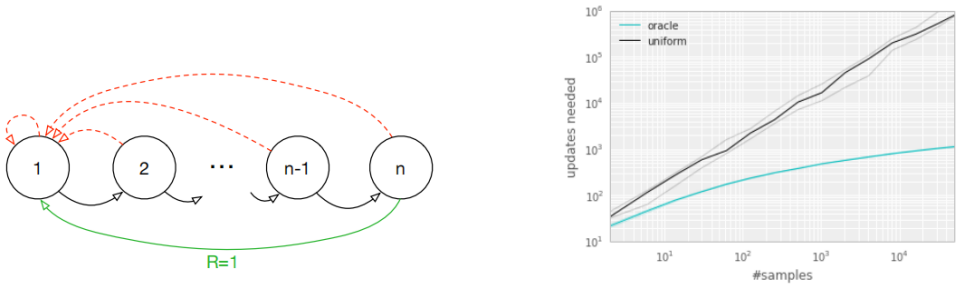


图 0.21　左侧：“盲走悬崖”示例的说明：存在 n 个状态，两种动作：“正确”动作和“错误”动作，每当智能体采取“错误”动作时（红色虚线箭头），幕就会终止。若一直采取“正确”动作会依次经过 n 个状态（黑色箭头），在这一序列的末尾有一个最终奖励 1（绿色箭头）；其他情况下奖励为 0。状态 1 为初始状态。我们采用的表征方式使得无法泛化出哪个动作是“正确”的。右侧：Q-学习为学习价值函数所需的更新次数的中位数，它是转移总数量的函数。注意使用的是双对数刻度，这突出了与均匀回放（黑色）相比，通过“oracle”回放（亮蓝色）所带来的指数级加速；浅色线条是来自 10 次独立运行的最小/最大值。

3.2 用 TD 误差进行优先级排序（贪婪）

优先级回放的核心部分是衡量每个转移重要性的准则。一种理想化的准则是 RL 智能体在当前状态下能从一个转移中学习到的知识量（预期学习进展）。虽然这种度量无法直接获取，但一个合理的替代指标是转移的 TD 误差 δ 的大小，它表明了该转移有多“令人惊讶”或出乎意料：具体而言，就是该状态的价值与基于下一步自举估计的价值之间的差距[11]。这种指标特别适用于增量式在线 RL 算法，例如 SARSA 或 Q-学习，这些算法本身就会计算 TD 误差，并按照与 δ 成比例的方式更新参数。在某些情况下，TD 误差也可能是一个较差的估计，例如当奖励存在噪声时；有关替代方案的讨论，请参见附录 A。

为了证明通过 TD 误差对回放进行优先级排序的潜在有效性，我们在“盲走悬崖”环境中，将均匀回放和 oracle 回放这两种基准情况，与一种“贪婪 TD 误差优先级排序”算法进行了比较。该算法会在回放记忆池中为每个转移存储最近一次遇到的 TD 误差。具有最大绝对 TD 误差的转移会在回放记忆池中回放，对这些转移应用 Q-学习更新，按照与 TD 误差成比例的方式更新权重。新的转移没有已知的 TD 误差，因此我们将它们设为最高优先级，以确保所有经验至少被看到一次。图 2（左侧）显示，该算法能大幅减少解决“盲走悬崖”任务所需的工作量[4]。

实现：为了适应大规模的记忆容量 N，我们使用二进制堆数据结构作为优先级队列。使用该结构时，采样时查找具有最大优先级的转移的时间复杂度为 $O(1)$，而更新优先级（在学习步骤后使用新的 TD 误差）的时间复杂度为 $O(\log N)$。更多细节请参见附录 B.2.1。

[4]需要注意的是，对于贪婪优先级排序，对 Q 值的进行随机（或乐观）初始化是必要的。如果改用零初始化，无奖励的转移最初会表现为误差为零，被置于队列的末尾，并且直到其他转移的误差降到数值精度以下时才会被重新访问。

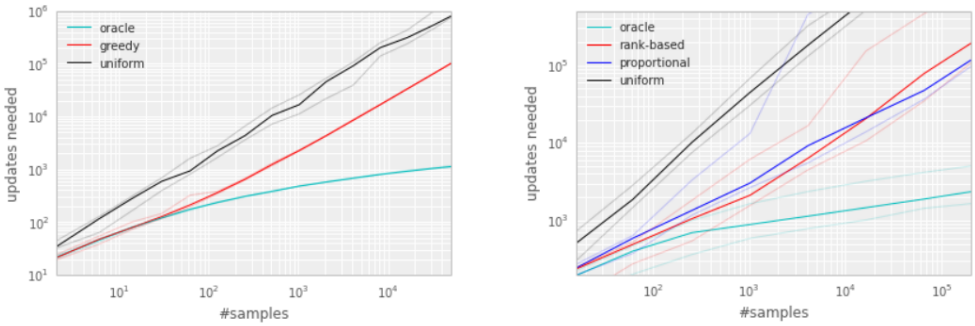


图 0.22　在“盲走悬崖”示例中，Q-学习为学习价值函数所需的更新次数的中位数，它是转移总数量的函数（其中只有一次转移是成功的，且获得了非零奖励）。浅色线条是来自 10 次随机初始化的最小/最大值。黑色代表均匀随机回放，青色使用 hindsight-oracle 来选择转移，红色和蓝色分别使用优先级回放（基于排名的和成比例的）。结果相差多个数量级，因此需要使用双对数图。在两个子图中，显然以正确的顺序回放经验，对所需的更新次数有极大的影响。有关详细信息，请参见附录 B.1。左侧：表格形式，贪婪优先级排序。右侧：线性函数逼近，两种随机优先级排序变体。

3.3 随机优先级排序

然而，贪婪的 TD 误差优先级排序存在几个问题。首先，为了避免对整个回放记忆池进行耗时的全面扫描，仅对被回放的转移的 TD 误差进行更新。由此带来的一个后果是，首次访问时 TD 误差较低的转移可能很长时间都不会被回放（对于滑动窗口式回放记忆池而言，这实际上意味着永远不会被回放）。（转移的 TD 误差对应的 critic 是上次取到该转移时的 critic，而不是当前的 critic。也就是说某一个转移的 TD-误差较低，只能够说明它对之前的 critic 的“信息量”不大，而不能说明它对当前的 critic 的“信息量”不大，因此根据 TD-误差进行贪婪选择有可能会错过对当前 critic“信息量”大的转移。）此外，它对噪声尖峰（例如奖励具有随机性时）较为敏感，而自举法会加剧这种敏感性——在自举法中，近似误差会成为另一噪声来源。最后，贪婪优先级排序会过度聚焦于经验的一小部分：误差衰减速度缓慢（尤其在使用函数逼近时），这意味着初始误差较高的转移会被频繁回放。这种多样性的缺失会导致系统容易出现过拟合问题。

为了克服这些问题，我们提出了一种介于纯贪心优先级排序与均匀随机采样之间的随机采样方法。我们确保被采样的概率随转移的优先级单调变化，同时保证即使是优先级最低的转移也有非零的被采样概率（这样能在充分利用高信息量经验的同时保证采样的多样性，避免过拟合）。具体而言，我们将采样转移 i 的概率定义为：

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{1}$$

其中 $p_i > 0$ 是转移 i 的优先级。指数 α 决定了优先级排序的使用程度，当 $\alpha = 0$ 时，对应均匀采样的情况。

我们考虑的第一种变体是直接的、成比例的优先级排序（定量），其中 $p_i = |\delta_i| + \epsilon$，这里 ϵ 是一个小的正常数，用于防止出现这样的极端情况：当转移的误差为零时，就不再被重新访问。第二种变体是间接的、基于排名的优先级排序（定性），其中 $p_i = \frac{1}{\text{rank}(i)}$，这里 $\text{rank}(i)$ 是当回放记忆池按照 $|\delta_i|$ 排序时转移 i 的排名。在这种情况下，P 成为一个指数为 α 的幂律分布。两种分布都随 $|\delta|$ 单调变化，但后者可能更鲁棒，因为它对异常值不敏感。如图 2（右侧）所示，在“盲走悬崖”任务中，两种随机优先级排序变体都比均匀采样的基准方法带来了大幅的速度提升。

实现：为了从分布 (1) 中高效采样，复杂度不能依赖于 N。对于基于排名的变体，我们可以用具有 k 个等概率分段的分段线性函数来近似累积密度函数。分段边界可以预先计算（只有当 N 或 α 变化时才会改变）。在运行时，我们先采样一个分段，然后在该分段内的转移中均匀采样。这种方法与基于小批量的学习算法结合时效果特别好：将 k 设为小批量的大小，然后从每个分段中恰好采样一个转移——这是一种分层采样形式，还具有平衡小批量的额外优势（将始终恰好有一个具有高幅度 δ 的转移、一个具有中等幅度的转移，等等）。（首先根据 TD 误差的绝对值利用二进制堆结构（完全二叉树）确定转移的“相对优先级排名”，然后计算转移的优先级概率，用分段线性函数近似累积密度函数，将所有经验按“优先级概率”划分为 k 个概率和相等的段（每一段的 $\sum P(i) = 1/k$），再从各段抽样。）成比例的变体则不同，但也允许基于“sum tree”数据结构（其中每个节点是其子节点的和，优先级作为叶节点）的高效实现，该结构可以高效地进行更新和采样。有关更多额外细节，请参见附录 B.2.1。

3.4 偏差退火

重要性采样：计算目标分布 $P_A(x)$ 下函数 $f(x)$ 的期望 $\mathbb{E}_{x\sim A}[f(x)]$，但存在以下问题：目标分布 $P_A(x)$ 难以直接采样（例如 $P_A(x)$ 的表达式复杂，或归一化常数未知），或目标分布 $P_A(x)$ 采样效率低（例如 $P_A(x)$ 是“长尾分布”，关键区域的样本极少，直接采样会导致估计偏差）。此时，重要性采样的思路是：找一个“易于采样的参考分布 $P_B(x)$”，对 $P_B(x)$ 采样，再用“权重”修正采样偏差，间接估计目标期望。

$$\mathbb{E}_{x\sim A}[f(x)] = \sum_x P_A(x) f(x) = \sum_x P_B(x) \frac{P_A(x)}{P_B(x)} f(x) = \mathbb{E}_{x\sim B}\left[\frac{P_A(x)}{P_B(x)} f(x)\right].$$

DQN 中引入经验池是为了解决数据相关性，使数据（尽量）独立同分布，公式 (3) 给出了随机梯度上升的形式。但是使用优先经验回放又改变了数据所满足的分布，即回放缓冲区的采样分布是优先级分布 $P(i)$，但训练时的目标分布是均匀分布 $U(i)$，分布不匹配会导致训练偏差，需用重要性采样修正梯度，即对“被过度采样的高

优先级样本”降权，对“被少采的低优先级样本”加权，确保损失函数的期望仍符合目标分布。

通过随机更新来估计期望值，依赖于这些更新与期望所对应的分布相同。优先级回放会引入偏差，因为它以不受控的方式改变了这个分布，因此也改变了估计值会收敛到的解（即使策略和状态分布是固定的）。我们可以通过使用重要性采样（IS）权重

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}$$

来纠正这种偏差，当 $\beta = 1$ 时，该权重能完全补偿非均匀概率 $P(i)$（(s,a) 是以 $P(i)$ 的概率选出来的，此时 w_i 和 $P(i)$ 抵消）。这些权重可以通过使用 $w_i\delta_i$ 而非 δ_i，融入到 Q-学习的更新中（因此这是加权重要性采样，而非普通的重要性采样，例如可参见[30]）。出于稳定性考虑，我们总是通过 $1/\max_i w_i$ 对权重进行归一化，使得它们仅对更新起到向下缩放的作用（控制极端值，稳定更新幅度，减小方差。归一化后的 w_i 在编写代码时可推导转化为下式）。

$$v_i = \frac{(N * P(i))^{-\beta}}{\max_k(w_k)} = \frac{(N * P(i))^{-\beta}}{\max_k((N * P(k))^{-\beta})} = \frac{(P(i))^{-\beta}}{\max_k((P(k))^{-\beta})} = \left(\frac{P(i)}{\min_k P(k)}\right)^{-\beta}.$$

在典型的强化学习场景中，更新的无偏特性在训练末期接近收敛时最为重要，因为由于策略、状态分布和自举目标的变化，整个过程本来就具有高度的非平稳性；我们假设在这种情况下，小的偏差是可以忽略的（另见附录中的图 12，该图是对雅达利平台上完全重要性采样校正的案例研究）。因此，我们利用了随时间退火重要性采样校正量的灵活性，通过为指数 β 定义一个调度表，使其仅在学习末期才达到 1。在实践中，我们将 β 从其初始值 β_0 线性退火到 1。需要注意的是，该超参数的选择与优先级排序指数 α 的选择相互影响；同时增大这两个参数，会在更强烈地校正偏差的同时，也更积极地优先进行采样。

在非线性函数逼近（例如深度神经网络）的背景下，当重要性采样与优先级回放结合时，还有另一项益处：在此场景中，大的步长可能极具破坏性，因为梯度的一阶近似仅在局部可靠，必须通过更小的全局步长来防止不良影响。在我们的方法中，优先级排序确保高误差的转移被多次看到，同时重要性采样校正会减小梯度的幅度（从而减小参数空间中的有效步长），并允许算法适应高度非线性的优化 landscape 的曲率，因为泰勒展开会被持续重新近似。

我们将优先级回放算法整合到一个全面的 RL 智能体中，该智能体基于最先进的 Double DQN 算法。我们的主要修改是，用我们的随机优先级排序和重要性采样方法，替代 Double DQN 所使用的均匀随机采样（参见算法 1）。

算法 3 采用成比例优先级排序的 Double DQN

输入： 小批量大小 k，步长 η，回放周期 K 与大小 N，指数 α 和 β，预算 T。

初始化回放记忆池 $\mathcal{H} = \emptyset$，$\Delta = 0$，$p_1 = 1$
观测 S_0 并选择动作 $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
    观测 S_t, R_t, γ_t
    将转移 $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ 存入 $\mathcal{H}$，并赋予最大优先级 $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ **then**
        for $j = 1$ **to** k **do**
            采样转移 $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            计算重要性采样权重 $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
            计算 TD 误差 $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
            更新转移优先级 $p_j \leftarrow |\delta_j|$
            累积权重变化 $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
        end for
        更新权重 $\theta \leftarrow \theta + \eta \cdot \Delta$，重置 $\Delta = 0$
        不时地将权重复制到目标网络 $\theta_{\text{target}} \leftarrow \theta$
    end if
    选择动作 $A_t \sim \pi_\theta(S_t)$
end for

4 雅达利实验

在明确所有这些概念后，我们现在研究采用这种优先级采样的回放能在多大程度上提升真实问题领域中的性能。为此，我们选择了雅达利基准测试集[31]，及其端到端的视觉 RL 框架，该测试集因其广泛适用性而备受青睐，包含延迟信用分配、部分可观测性以及函数逼近难度较大等多样化挑战[3,23]。我们的假设是，优先级回放通常是有用的，能够在无需针对具体问题进行超参数精细调校的情况下，显著提升经验回放的学习效率。

我们考虑了两种使用均匀经验回放的基准算法，即 Nature 论文中的 DQN 算法[3]，以及其近期扩展的 Double DQN 算法[23]。后者通过引入 Double Q-学习机制[32]，有效降低了高估计偏差，显著提升了算法性能。本文采用优化后的 Double DQN 算法版本。这些基准算法的核心在于其经验回放机制：所有历史状态转移数据均存储在滑动窗口记忆池中，该回放记忆池仅保留最近 10^6 个转移。算法会从回放记忆池中均匀采样 32 个转移组成小批量数据进行处理，每当有 4 个新转移进入记忆池时，就会执行一次小批量更新，因此平均每个转移数据会被回放 8 次。为确保稳定性，奖励和 TD 误差均经过截断处理，使其落在 $[-1, 1]$ 范围内。

我们使用与基准算法完全相同的神经网络架构、学习算法、回放记忆池和评估设置（见附录 B.2）。唯一的区别在于从回放记忆池中采样转移的机制，现在是

根据算法 1 进行采样，而非均匀采样。我们将基准算法与优先级回放的两种变体（基于排名的和成比例的）进行比较。

与基准算法相比，仅需要进行一项超参数调整：由于优先级回放更频繁地选取高误差的转移，梯度幅度会更大，因此我们将步长 η 相较于（Double）DQN 的设置缩小了 4 倍。对于由优先级排序引入的 α 和 β_0 超参数，我们进行了粗略的网格搜索（在 8 款游戏的子集上评估），发现基于排名的变体的最佳点为 $\alpha = 0.7$、$\beta_0 = 0.5$，而成比例的变体的最佳点为 $\alpha = 0.6$、$\beta_0 = 0.4$。这些选择在激进性和鲁棒性之间进行了权衡，但通过减小 α 和/或增大 β，很容易恢复到更接近基准算法的行为。

我们通过在所有游戏上使用单一超参数设置运行每种变体来生成结果，就像基准算法所做的那样。我们的主要评估指标是最佳策略的质量，具体表现为幕的平均得分。这里的起始状态是从人类轨迹中采样而来的（这一设定由[21] 提出，并在[23] 的研究中使用，这种设定要求智能体具备更强的鲁棒性和泛化能力，因为智能体无法依赖重复单一记忆的轨迹。）这些结果汇总在表 1 和图 3 中，但完整结果和原始得分可在附录的表 7 和表 6 中找到。次要指标是学习速度，我们在图 4 中对其进行了汇总，更详细的学习曲线在图 7 和图 8 中。

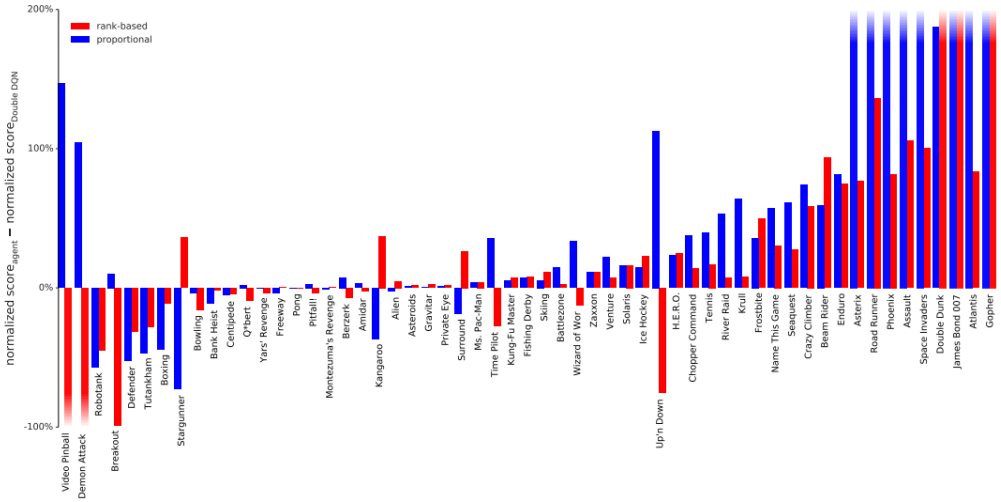


图 0.23　在 57 款采用“人类起始”设定的游戏中，无优先级回放与含有优先级回放的 Double DQN 的标准化得分差异（以“随机水平”与“人类水平”的差距定义为 100% 作为基准）。其中，红色代表基于排名的优先级回放变体，蓝色代表成比例的优先级回放变体。结果显示，在大多数游戏中，加入优先级回放的 Double DQN 均实现了显著性能提升。具体得分详见表 6，另可参考图 9（该图以常规 DQN 为基准）。

我们发现，在 DQN 中加入优先级回放后，49 款游戏中有 41 款的得分得到了显著提升（对比附录中表 6 的第 2 列和第 3 列，或图 9），49 款游戏的标准化性能中位数从 48% 提升至 106%。此外，我们发现优先级经验回放带来的性能提升，与在 DQN 中引入 Double Q-学习带来的提升是互补的：性能又上了一个台阶，从

	DQN		Double DQN (tuned)		
	baseline	rank-based	baseline	rank-based	proportional
Median	48%	106%	111%	113%	128%
Mean	122%	355%	418%	454%	551%
> baseline	–	41	–	38	42
> human	15	25	30	33	33
# games	49	49	57	57	57

表 10　标准化得分汇总。完整结果见附录表 6。

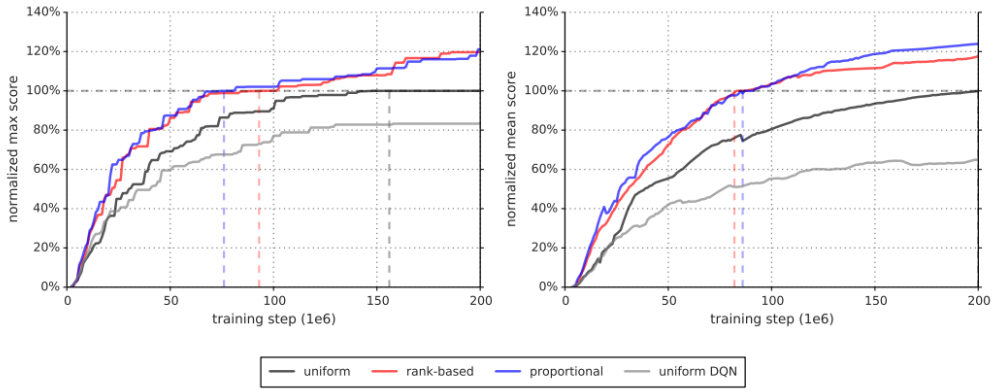


图 0.24　学习速度汇总图。左侧：在 57 款游戏上，截至目前所达到的、经基准标准化的最高得分的中位数。基线归一化得分的计算方式如公式 (4) 所示，但使用训练过程中观察到的 Double DQN 最高得分代替人类得分。等效点用虚线突出显示；这些是曲线达到 100% 时的步数（即，此时算法在游戏的中位数表现方面与 Double DQN 相当）。对于基于排名和成比例的优先级排序，这些等价点分别出现在总训练时间的 47% 和 38% 处。右侧：与左侧类似，但使用均值而非最大值，这捕捉的是累积性能而非峰值性能。在此，基于排名和成比例的优先级排序分别在总训练时间的 41% 和 43% 处达到等效点。有关这些图所汇总的详细学习曲线，请参见图 7。

而在雅达利基准测试中达到了当前的最先进水平（见图 3）。与 Double DQN 相比，57 款游戏的性能中位数从 111% 提升至 128%，平均性能从 418% 提升至 551%，这使得 River Raid，Seaquest 和 Surround 等游戏首次达到了人类水平，同时在其他游戏（如 Gopher，James Bond 007 或 Space Invaders）上也有大幅提升。需要注意的是，平均性能并非一个非常可靠的指标，因为有一款游戏（Video Pinball）的影响占主导地位。优先级回放几乎在所有游戏上都带来了性能提升，总体而言，学习速度提高了一倍（见图 4 和图 8）。图 7 中的学习曲线表明，虽然两种优先级排序变体通常会产生相似的结果，但在某些游戏中，其中一种变体的表现接近 Double DQN 基准，而另一种则带来了大幅提升，例如，基于排名的变体在 Double Dunk 或 Surround 中，成比例的变体在 Alien，Asterix，Enduro，Phoenix 或 Space Invaders 中。从学习曲线中还可以观察到，与均匀基准相比，等存在明显延迟问题的游戏中（如 Battlezone，Zaxxon 或 Frostbite）中，优先级排序能有效地减少性能起步前的延迟。

5 讨论

在基于排名的优先级排序和成比例的优先级排序的直接对比中，我们曾预期基于排名的变体更鲁棒，因为它不受异常值或误差幅度的影响。此外，它的重尾特性还能保证样本的多样性，而且从不同误差的分区进行分层采样，会在整个训练过程中使小批量的总梯度保持在稳定的幅度。另一方面，排名会让算法对相对误差尺度不敏感，当存在可利用的误差分布结构时（例如在稀疏奖励场景中），这可能会导致性能下降。或许令人惊讶的是，这两种变体在实践中的表现相似；我们推测这是由于 DQN 算法中大量使用了（对奖励和 TD 误差的）截断操作，从而消除了异常值。我们监测了多款游戏中 TD 误差随时间的分布情况（见附录中的图 10），发现随着学习的推进，它会接近重尾分布，同时在不同游戏间仍存在显著差异；这从经验上验证了公式 (1) 的形式。附录中的图 11 展示了该分布与公式 (1) 的相互作用，从而产生有效回放概率的过程。

在进行这项分析时，我们偶然发现了另一种现象（事后看来很明显），即部分已访问的转移在从滑动窗口记忆池中被淘汰前，从未被回放过，还有更多转移在首次遇到很久之后才被首次回放。此外，均匀采样会隐性地偏向于过时的转移，这些转移是由一个自生成以来通常已经历了数十万次更新的策略所产生的。优先级回放机制通过为未见过的转移提供更高的优先级，直接纠正了第一个问题，也往往有助于解决第二个问题，因为较新的转移往往具有更大的误差——这是因为旧转移有更多机会被修正，而且价值函数对新数据的预测往往更不准确。

我们推测深度神经网络与优先级回放还存在另一种有趣的交互方式。当我们把学习给定表征下的价值（即顶层网络）和学习改进的表征（即底层网络）区分开时，那些表征良好的转移会快速降低其误差，进而被回放的次数大幅减少，这就使得学习重点更多地放在表征欠佳的其他转移上，从而在观测和网络容量允许的情况下，将更多资源投入到区分混淆状态上。

6 扩展

优先级监督学习：在监督学习的背景下，与优先级回放类似的方法是从数据集中进行非均匀采样，每个样本都基于其最近的误差来确定优先级。这有助于将学习重点放在仍能从中学习的样本上，为（困难的）边界情况投入额外资源，这与提升方法（boosting）[16] 有些类似。此外，如果数据集存在不平衡问题，我们推测稀有类别的样本会被以不成比例的频率采样，因为它们的误差缩小得更慢，而从常见类别中选择的样本将是那些最接近决策边界的样本，从而产生类似于硬负例挖掘[33] 的效果。为了验证这些直觉是否成立，我们在经典 MNIST 数字分类问题[34]

的类不平衡变体上进行了初步实验：我们移除了训练集中数字 0、1、2、3、4 的 99% 样本，而保持测试/验证集不变（即这些集合仍保持类平衡）。我们比较了两种场景：在“知情”场景中，我们人为地对样本量不足的类别误差进行加权（乘以系数 100）；在“不知情”场景中，我们不提示测试分布会与训练分布不同。有关卷积神经网络训练设置的详细信息，请参见附录 B.3。优先级采样（不知情，$\alpha = 1$，$\beta = 0$）优于不知情的均匀基准，并且在泛化方面接近知情的均匀基准（见图 5）；同样，优先训练在学习速度方面也更快。

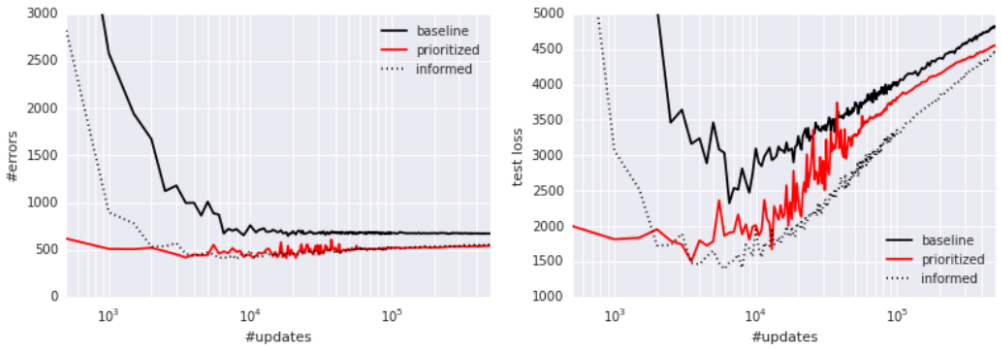


图 0.25　在严重类不平衡的 MNIST 数据集上，分类误差随监督学习更新次数的变化。优先级采样提升了性能，导致测试集上的误差相当，并接近先验已知不平衡信息的性能（随机初始化的中位数为 3）。左侧：测试集分类错误样本数。右侧：测试集损失，突出过拟合问题。

Off-policy 回放：Off-policy RL 的两种标准方法是拒绝采样（目标分布为 $p(x)$，提议分布为 $q(x)$，满足 $p(x) \le Mq(x)$，从 $q(x)$ 中生成一个样本 x_i，从 $[0,1]$ 上的均匀分布中生成一个随机数 u，计算接受概率 $\alpha(x_i) = p(x_i)/(Mq(x_i))$，若 $u \le \alpha(x_i)$，则接受样本 x_i 作为来自目标分布为 $p(x)$ 的一个样本，否则则拒绝），以及使用重要性采样比率 ρ 来校正转移在 on-policy 情况下发生的可能性。我们的方法包含了与这两种方法类似的部分，即回放概率 P 和重要性采样校正项 w。因此，如果转移可在回放记忆池中获取，将我们的方法应用于 off-policy RL 似乎是很自然的。具体而言，在成比例的变体中，当 $w = \rho$、$\alpha = 0$、$\beta = 1$ 时（均匀回放），我们得到加权重要性采样；当 $p = \min(1, \rho)$、$\alpha = 1$、$\beta = 0$ 时（$\rho = p(x)/q(x)$，$M = \sum_k P(k)$），得到拒绝采样。我们的实验表明，可能带有退火或排名机制的中间变体在实践中可能更有用——尤其是当重要性采样比率引入高方差时，即当目标策略在某些状态下与行为策略存在显著差异时。当然，off-policy 校正与我们基于预期学习过程的优先级排序是互补的，并且可以使用相同的框架进行混合优先级排序，例如通过定义 $p = \rho \cdot |\delta|$，或者基于 ρ 和 δ 进行其他合理的权衡（ρ 修正偏差，δ 提升效率，既重视策略差异小的经验，有重视学习价值高的经验）。

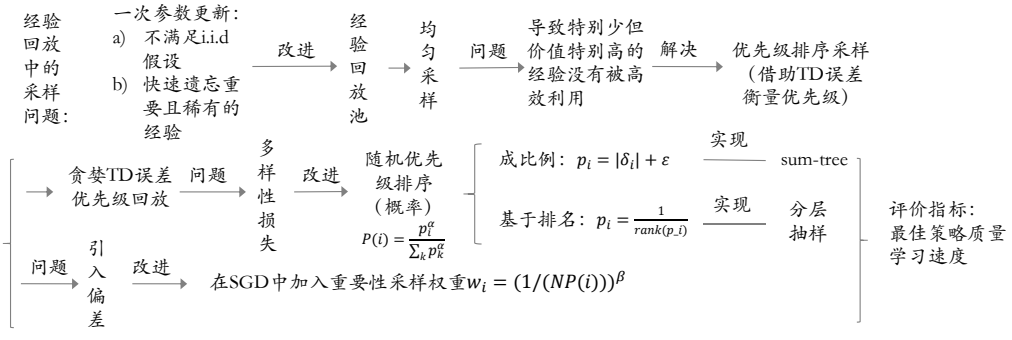
探索的反馈：优先级回放的一个有趣副作用是，一个转移最终被回放的总次数 M_i 差异很大，而这粗略地表明了它对智能体的有用程度。这种潜在有价值的信

号可以反馈给生成转移的探索策略中。例如，我们可以在每一幕开始时，从一个参数化的分布中采样探索时用到的超参数（如随机动作的比例 ϵ、玻尔兹曼温度，或混合时需要加入的内在奖励的量），通过 M_i 监测经验的有用性，然后更新该分布以生成更有用的经验。或者，在类似 Gorila 智能体[21] 的并行系统中，它可以指导在一组并发但异构的“actors”之间进行资源分配，每个 actor 具有不同的探索超参数。

优先级记忆：在决定哪些转移需要回放时，相关考量同样适用于确定哪些转移值得保留以及何时删除（例如当这些记忆很可能永远不会再被回放时）。通过明确控制保留或删除哪些转移，可以有效缩减所需存储总量——既减少了冗余（高频访问的转移的误差会很低，因此其中很多会被丢弃），同时会自动根据已学到的内容进行调整（丢弃很多“简单”的转移），并使内存内容偏向于误差仍较高的部分。这是一个非平凡（重要且具挑战性）的问题，因为当前 DQN 的内存需求主要由回放记忆池的规模大小决定，而非神经网络本身的大小。删除是比降低回放概率更彻底的决定，因此可能需要更加强调经验的多样性，例如通过跟踪每个转移的时效性，并以此来调整优先级，从而保留足够的旧经验以防止循环（这与多智能体文献中的“hall of fame”理念相关[35]）。优先级机制也足够灵活，允许整合其他来源的经验，比如来自规划器或人类专家轨迹的经验[19]，因为明确经验来源有助于调节每个动作的优先级，例如通过这种方式在回放记忆池中保留足够比例的外部经验。

7 结论

本文提出了一种名为优先级回放的创新方法，该技术能显著提升经验回放的学习效率。我们深入研究了几种变体，设计了可扩展到大型回放记忆池的实现方式，并且发现优先级回放使学习速度提升两倍，还在雅达利基准测试中创下性能新高。此外，我们还提出了其他有前景的变体和扩展，特别是针对类别不平衡监督学习场景的解决方案。



参考文献

[1] Lin, Long-Ji. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning, 8(3-4):293–321, 1992.

[2] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[3] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[4] Schmidhuber, Jurgen. Curious model-building control systems. In Neural Networks, 1991. 1991 IEEE International Joint Conference on, pp. 1458–1463. IEEE, 1991.

[5] Atherton, Laura A, Dupret, David, and Mellor, Jack R. Memory trace replay: the shaping of memory consolidation by neuromodulation. Trends in neurosciences, 38(9):560–570, 2015.

[6] Olafsdottir, H Freyja, Barry, Caswell, Saleem, Aman B, Hassabis, Demis, and Spiers, Hugo J. Hippocampal place cells construct reward related sequences through unexplored space. Elife, 4: e06063, 2015.

[7] Foster, David J and Wilson, Matthew A. Reverse replay of behavioural sequences in hippocampal place cells during the awake state. Nature, 440(7084):680–683, 2006.

[8] Singer, Annabelle C and Frank, Loren M. Rewarded outcomes enhance reactivation of experience in the hippocampus. Neuron, 64(6):910–921, 2009

[9] McNamara, Colin G, Tejero-Cantero, Alvaro, Trouche, Stephanie, Campo-Urriza, Natalia, and Dupret, David. Dopaminergic neurons promote hippocampal reactivation and spatial memory persistence. Nature neuroscience, 2014.

[10] Moore, Andrew W and Atkeson, Christopher G. Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning, 13(1):103–130, 1993.

[11] Andre, David, Friedman, Nir, and Parr, Ronald. Generalized prioritized sweeping. In Advances in Neural Information Processing Systems. Citeseer, 1998.

[12] van Seijen, Harm and Sutton, Richard. Planning by prioritized sweeping with small backups. In Proceedings of The 30th International Conference on Machine Learning, pp. 361–369, 2013.

[13] White, Adam, Modayil, Joseph, and Sutton, Richard S. Surprise and curiosity for big data robotics. In Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014.

[14] Geramifard, Alborz, Doshi, Finale, Redding, Joshua, Roy, Nicholas, and How, Jonathan. Online discovery of feature dependencies. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 881–888, 2011.

[15] Sun, Yi, Ring, Mark, Schmidhuber, Jurgen, and Gomez, Faustino J. Incremental basis construction from temporal difference error. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 481–488, 2011.

[16] Galar, Mikel, Fernandez, Alberto, Barrenechea, Edurne, Bustince, Humberto, and Herrera, Francisco. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 42(4):463–484, 2012.

[17] Narasimhan, Karthik, Kulkarni, Tejas, and Barzilay, Regina. Language understanding for text-based games using deep reinforcement learning. In Conference on Empirical Methods in Natural Language Processing (EMNLP), 2015.

[18] Hinton, Geoffrey E. To recognize shapes, first learn to generate images. Progress in brain research, 165:535–547, 2007.

[19] Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), Advances in Neural Information Processing Systems 27, pp. 3338–3346. Curran Associates, Inc., 2014.

[20] Stadie, Bradly C, Levine, Sergey, and Abbeel, Pieter. Incentivizing exploration in reinforcement learning with deep predictive models. arXiv preprint arXiv:1507.00814, 2015.

[21] Nair, Arun, Srinivasan, Praveen, Blackwell, Sam, Alcicek, Cagdas, Fearon, Rory, Maria, Alessandro De, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, Legg, Shane, Mnih, Volodymyr, Kavukcuoglu, Koray, and Silver, David. Massively parallel methods for deep reinforcement learning. arXiv preprint arXiv:1507.04296, 2015.

[22] Bellemare, Marc G., Ostrovski, Georg, Guez, Arthur, Thomas, Philip S., and Munos, Remi. Increasing the action gap: New operators for reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence, 2016. URL http://arxiv.org/abs/1512.04860.

[23] van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep Reinforcement Learning with Double Q-learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016. URL http://arxiv.org/abs/1509.06461.

[24] Wang, Z., de Freitas, N., and Lanctot, M. Dueling network architectures for deep reinforcement learning. Technical report, 2015. URL http://arxiv.org/abs/1511.06581.

[25] Watkins, Christopher JCH and Dayan, Peter. Q-learning. Machine learning, 8(3-4):279–292, 1992.

[26] Schaul, Tom, Zhang, Sixin, and Lecun, Yann. No more pesky learning rates. In Proceedings of the 30th International Conference on Machine Learning (ICML-13), pp. 343–351, 2013.

[27] Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2014.

[28] Diamond, Jared. Zebras and the Anna Karenina principle. Natural History, 103:4–4, 1994.

[29] Riedmiller, Martin. Rprop-description and implementation details. 1994.

[30] Mahmood, A Rupam, van Hasselt, Hado P, and Sutton, Richard S. Weighted importance sampling for off-policy learning with linear function approximation. In Advances in Neural Information Processing Systems, pp. 3014–3022, 2014.

[31] Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. arXiv preprint arXiv:1207.4708, 2012.

[32] van Hasselt, Hado. Double Q-learning. In Advances in Neural Information Processing Systems, pp.2613–2621, 2010.

[33] Felzenszwalb, Pedro, McAllester, David, and Ramanan, Deva. A discriminatively trained, multiscale, deformable part model. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pp. 1–8. IEEE, 2008.

[34] LeCun, Yann, Cortes, Corinna, and Burges, Christopher JC. The MNIST database of handwritten digits, 1998.

[35] Rosin, Christopher D and Belew, Richard K. New methods for competitive coevolution. Evolutionary Computation, 5(1):1–29, 1997.

[36] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[37] Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clement. Torch7: A matlab-like environment for machine learning. In BigLearn, NIPS Workshop, number EPFL-CONF-192376, 2011.

A 优先级排序变体

绝对 TD 误差只是期望学习进展的理想优先级度量的一种可能替代指标。虽然它捕捉了潜在改进的规模，但忽略了奖励或转移中的固有随机性（此时若一个转移的 TD 误差大可能是运气所致，而非真的需要学习），以及部分可观测性或函数逼近（FA）能力的可能限制（模型表达能力不足，TD 误差大但学不会）；换句话说，当存在不可学习的转移时，它会存在问题。在这种情况下，它的导数——可以通过某一转移当前的 $|\delta|$ 与其上次回放时的 $|\delta|$ 之间的差值来近似（变化大的对学习的边际贡献大，值得优先回放）[5] ——可能更有用。不过，这种度量并非立即可得，而且会受到其间回放内容的影响，这增加了它的方差。在初步实验中，我们发现它的表现并未超过 $|\delta|$，但这可能更多地反映了我们所研究的（接近确定性的）环境类别，而非该度量本身的问题。

一种正交的变体是考虑回放一个转移所引发的权重变化的范数——如果底层优化器采用自适应步长，以降低高噪声方向上的梯度[26,27]，那么这可能会很有效，从而将区分可学习和不可学习转移的负担交给优化器。

我们可以通过差异化处理正 TD 误差和负 TD 误差来调节优先级排序机制；例如，我们可以引用 Anna Karenina 原则[28]，其含义是，有很多种方式能让转移比预期差（比如环境随机、策略错误、价值估计本身有偏差等），只有一种方式能让转移比预期好（即实际回报确实更高），据此引入一种不对称性，对幅度相等的正 TD 误差比对负 TD 误差给予更高的优先级，因为前者更有可能提供信息。在大鼠研究中也观察到了这种回放频率的不对称性[8]。同样，我们对这类变体的初步实验结果尚无定论。

神经科学的证据表明，基于幕回报而非预期学习进展的优先级排序可能也有用[5–7]。在这种情况下，我们可以提高整幕的回放概率，而非单个转移的回放概率，或者根据观测到的未来回报（甚至它们的价值估计）来提高单个转移的优先级。

对于保持足够多样性（以防止过拟合、过早收敛或表征匮乏）的问题，除了我们选择引入随机性的方法外，还有其他解决方案，例如，优先级可以通过观测空间中的新颖性度量来调整。完全可行的是采用混合策略：（每个小批量的）一部分元素根据一种优先级度量采样，其余部分根据另一种优先级度量采样，从而引入额外的多样性。另一种正交的想法是，通过引入显式的“陈旧性奖励”来提高一段时间未被回放的转移的优先级，以确保每个转移都能不时地被重新访问，且其被回访的概率会随着最后一次 TD 误差变得陈旧的速率同步递增。在这种奖励随时间线性增长的简单情况下，这可以在不增加额外成本的情况下实现，方法是在每次

[5] 当然，更鲁棒的近似应该是所有遇到的 δ 值的历史的函数。特别是，可以想象一种类似 RProp 风格的更新[29] 来处理优先级：当符号匹配时提高优先级，而每当连续的误差（对于同一转移）符号不同时降低优先级。

更新时从新优先级中减去一个与全局步数成比例的量[6]。

在基于价值函数进行自举的 RL 的特定情形中，有可能可以利用回放记忆池的序列结构，其依据如下直觉：若一个转移带来了大量学习（对后继状态的价值估计有显著的改进），则这个转移有潜力改变所有进入该状后继态的转移的自举目标（因为自举的目标值依赖该后继状态的价值估计），因此关于这些转移还有更多可学习的内容。当然，我们至少知道其中一个，即历史上的前驱转移，因此提高它的优先级会使其更有可能很快被重新访问。与资格迹类似，让“未来的回报”反向传递到“导致该回报的动作和状态”的价值估计中。在实践中，我们将当前转移的 $|\delta|$ 添加到前驱转移的优先级中，但仅当前驱转移不是终止转移时。这个想法与在啮齿动物中观察到的“反向回放”[7] 以及优先扫除的近期扩展[12] 相关。

B 实验细节

B.1 盲走悬崖（Blind Cliffwalk）

对于盲走悬崖实验（3.1 节及后续内容），我们使用了直接的 Q 学习[25] 设置。Q 值要么用表格形式，要么用线性函数逼近器表示，在这两种情况下都表示为 $Q(s,a) := \theta^\top \phi(s,a)$。对于每个转移，我们使用以下公式计算其时序差分（TD）误差：

$$\delta_t := R_t + \gamma_t \max_a Q(S_t, a) - Q(S_{t-1}, A_{t-1}) \tag{2}$$

并使用随机梯度上升来更新参数：

$$\theta \leftarrow \theta + \eta \cdot \delta_t \cdot \nabla_\theta Q|_{S_{t-1}, A_{t-1}} = \theta + \eta \cdot \delta_t \cdot \phi(S_{t-1}, A_{t-1}) \tag{3}$$

对于线性函数逼近（FA）的情况，我们使用非常简单的状态编码，将其表示为 one-hot 向量（与表格型的情况类似），但与一个值为 1 的恒定偏置特征拼接在一起（额外的自由度，避免欠拟合）。为了使跨动作的泛化成为不可能，我们对每个状态交替设置哪个动作是“正确的”，哪个是“错误的”（每个动作的正确性依赖具体的状态，让模型无法通过对一个动作的学习，泛化到其他动作的价值估计）。所有元素都初始化为接近零的小值，$\theta_i \sim \mathcal{N}(0, 0.1)$。

我们将问题的规模（状态数量 n）从 2 变化到 16。折扣因子设置为 $\gamma = 1 - \frac{1}{n}$，这使得值在不依赖于 n 的情况下大致保持在相同的尺度上。这允许我们在所有实验中使用固定的步长 $\eta = \frac{1}{4}$。

回放记忆池是通过穷举执行所有 2^n 种可能的动作序列直至终止（以随机顺

[6]如果自举与策略迭代结合使用，使得目标值来自单独的网络（就像 DQN 的情况一样），那么当目标网络在外层迭代中更新时，关于优先级的不确定性会大幅增加。在这些时刻，陈旧性奖励会与中间发生的单独（低层级）更新的次数成比例地增加。

序）来填充的。这保证了恰好有一个序列会成功并获得最终奖励，而所有其他序列都会失败且奖励为零。回放记忆池包含了所有相关经验（转移的总数为 $2^{n+1}-2$），其出现频率与采用随机行为策略在线行动时遇到的频率一致。鉴于此，原则上我们只需增加计算量就能学习到收敛；在此，收敛被定义为估计的 Q 值与真实值之间的均方误差（MSE）低于 10^{-3}。

B.2 雅达利（Atari）实验

B.2.1 实现细节

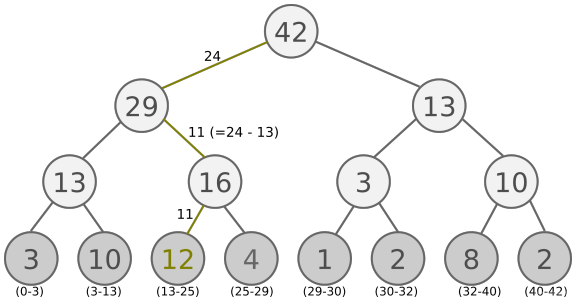
使用包含 $N = 10^6$ 个转移的回放记忆池进行优先级排序带来了一些性能挑战。在此，我们描述为最小化额外运行时间和内存开销所做的若干工作，以扩展第 3 节中的讨论。

基于排名的优先级排序. 雅达利的早期实验表明，维护一个包含 10^6 个转移且 TD 误差不断变化的排序数据结构，会主导运行时间。我们的最终解决方案是将转移存储在一个用基于数组的二叉堆实现的优先队列中。然后，直接将堆数组用作排序数组的近似，每 10^6 步对其进行一次不频繁的排序，以防止堆变得过于不平衡。这是对二叉堆的一种非常规使用，但我们在较小环境上的测试表明，与使用完美排序的数组相比，学习不受影响。这可能是因为上次看到的 TD 误差只是转移有用性的一个替代指标，而且我们使用了随机优先采样。运行时间的一个小改进来自避免对采样分布的分区进行过多的重新计算。对于接近的 N 值，我们重复使用相同的分区，并且不频繁地更新 α 和 β。我们基于排名的优先级排序的最终实现使运行时间额外增加了 2%-4%，且额外内存使用可忽略不计。这可以通过多种方式进一步减少，例如使用更高效的堆实现，但对于我们的实验来说已经足够好了。

成比例的优先级排序. 此处使用的“sum-tree”数据结构在本质上与二叉堆的数组表示非常相似。然而，与通常的堆性质不同，父节点的值是其子节点值的和。叶节点存储转移的优先级，内部节点是中间和，父节点包含所有优先级的总和 p_{total}。这提供了一种高效计算优先级累积和的方法，允许 $O(\log N)$ 的更新和采样操作。要采样一个大小为 k 的小批量，范围 $[0, p_{\text{total}}]$ 被等分为 k 个范围。

接下来，从每个范围中均匀采样一个值。最后，从树中检索与这些采样值对应的转移（从根节点开始往下找：对比当前节点的左子节点值，比左子节点大，就去右子节点，同时更新当前值（减去右子节点值）；否则去左子节点，直到找到叶子节点，对应的就是要采样的样本。例如，如果将所有节点的优先级加起来是 42，抽 6 个样本，此时的优先级区间划分为：[0-7]，[7-14]，[14-21]，[21-28]，[28-35]，[35-42]，然后在每个区间里随机选取一个数，比如在区间 [21-28] 里选到了 24，就按照这个 24 从最顶上的 42 开始向下搜索。首先看到 42 下面有两个子节点，用 24

对比左子节点 29，其小于 29，因此选择左边的路；29 下面有两个子节点，用 24 对比左子节点 13，其大于 13，因此选择右边的路，并且将 24 的值根据 13 进行修改，变为 24-13 = 11；13 下面有两个子节点，用 11 对比左子节点 12, 其小于 12，因此我们选择 12 对应的数据。）。开销与基于排名的优先级排序类似。



Sum-tree 示意图

如第 3.4 节所述，每当使用重要性采样时，所有权重 w_i 都会被缩放，使得 $\max_i w_i = 1$。我们发现在实践中这样效果更好，因为它将所有权重保持在合理范围内，避免了出现极大更新的可能性。值得一提的是，这种归一化与 β 的退火相互作用：当 β 接近 1 时，归一化常数会增大，这会以类似于退火步长 η 的方式降低有效平均更新。

B.2.2 超参数

在本文中，我们的基准是 DQN 和调优后的 Double DQN。我们在雅达利游戏的一个子集上对超参数进行了调优：Breakout，Pong，Ms. Pac-Man，Q*bert，Alien，Battlezone，Asterix。表 2 列出了尝试过的值，表 3 列出了选定的参数。

Hyperparameter	Range of values
α	0, 0.4, 0.5, 0.6, 0.7, 0.8
β	0, 0.4, 0.5, 0.6, 1
η	η_{baseline}, $\eta_{\text{baseline}}/2$, $\eta_{\text{baseline}}/4$, $\eta_{\text{baseline}}/8$

表 11　实验中考虑的超参数。此处 $\eta_{\text{baseline}} = 0.00025$。

	DQN		Double DQN (tuned)		
Hyperparameter	baseline	rank-based	baseline	rank-based	proportional
α (priority)	0	$0.5 \to 0$	0	0.7	0.6
β (IS)	0	0	0	$0.5 \to 1$	$0.4 \to 1$
η (step-size)	0.00025	$\eta_{\text{baseline}}/4$	0.00025	$\eta_{\text{baseline}}/4$	$\eta_{\text{baseline}}/4$

表 12　DQN 优先级变体的选定超参数。箭头表示线性退火，在训练结束时达到极限值。注意，以 DQN 为基准的基于排名的变体是没有重要性采样（IS）的早期版本。在此，优先级回放引入的偏差通过将 α 退火到零来纠正。

B.2.3 评估

两种评估方式：(a) 我们使用[23] 中描述的“人类起始”评估方法来评估我们的智能体。“人类起始”评估使用从人类轨迹中随机采样的起始状态。(b) 智能体在训练期间定期进行的 test 评估，使用的是在每一幕开始时执行随机数量的无操作（no - ops，即不进行有效动作，只让游戏空转）来随机化起始状态。“人类起始”评估的具体操作：在总计 30 分钟的游戏时间内，进行 100 次评估，然后对得分取平均。所有学习曲线图表显示的都是 test 评估下的得分，且这些图表是使用相同的代码库、相同的随机种子初始化生成的。

表 4 和表 5 展示了评估方法的差异，以及在评估期间每个智能体的 ϵ-贪婪策略中使用的 ϵ。使用“人类起始”评估的智能体是训练期间找到的最佳智能体，如[23] 所述。

Evaluation method	Frames	Emulator time	Number of evaluations	Agent start point
Human starts	108,000	30 mins	100	human starts
Test	500,000	139 mins	1	up to 30 random no-ops

表 13　评估方法比较。

	DQN		Double DQN (tuned)		
Evaluation method	baseline	rank-based	baseline	rank-based	proportional
Human starts	0.05	0.01	0.001	0.001	0.001
Test	0.05	0.05	0.01	0.01	0.01

表 14　每种评估方法下，每个智能体在 ϵ-贪婪策略中使用的 ϵ。

标准化得分的计算方式如[23] 所述：

$$\text{score}_{\text{normalized}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{|\text{score}_{\text{human}} - \text{score}_{\text{random}}|}, \tag{4}$$

注意分母取了绝对值。这仅会对 Video Pinball 产生影响，在该游戏中随机得分高于人类得分。结合较高的智能体得分，Video Pinball 对平均标准化得分有很大影响。我们继续使用这种（计算方式），以便我们的标准化得分具有可比性。

B.3 类不平衡的 MNIST

B.3.1 数据集设置

在我们的监督学习场景中，我们对 MNIST 进行了修改，以获得一个具有显著标签不平衡的新训练数据集。这个新数据集是通过选取对应前 5 个数字（0、1、2、3、4）的样本的一小部分，以及对应剩余 5 个标签（5、6、7、8、9）的所有样本来得到的。对于前 5 个数字中的每一个，我们随机抽取了可用样本的 1%，即可用 0

的 1%、可用 1 的 1% 等。在得到的数据集中，包含所有 10 个不同类别的样本，但它是高度不平衡的，因为对应 5、6、7、8、9 类的样本数量是 0、1、2、3、4 类样本数量的 100 倍。在我们所有的实验中，我们使用了原始的 MNIST 测试数据集，没有移除任何样本。

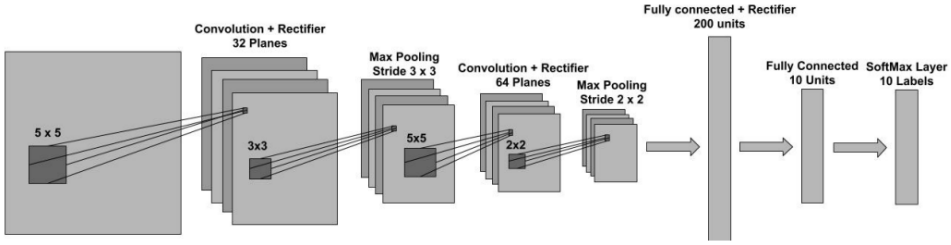


图 0.26　优先监督学习实验中使用的前馈网络架构。

B.3.2 训练设置

在我们的实验中，我们使用了一个 4 层前馈神经网络，其架构与 LeNet5[36] 类似。这是一个 2 层卷积神经网络，顶部接着 2 个全连接层。每个卷积层由纯卷积、整流非线性（ReLU）和下采样最大池化操作组成。网络中的两个全连接层也由整流非线性分隔。最后一层是 softmax 层，用于得到可能标签上的归一化分布。完整架构如图 6 所示，并使用 Torch7[37] 实现。模型使用随机梯度下降训练，无动量，小批量大小为 60。在我们所有的实验中，我们考虑了 6 种不同的步长（0.3、0.1、0.03、0.01、0.003 和 0.001），对于本文中呈现的每种情况，我们选择了能带来最佳（平衡的）验证性能的步长。我们使用负对数似然损失准则，并对损失的加权和未加权版本都进行了实验。在加权情况下，对应前 5 个数字（0、1、2、3、4）的样本的损失被乘以 100，以适应上述训练集中的标签不平衡情况。

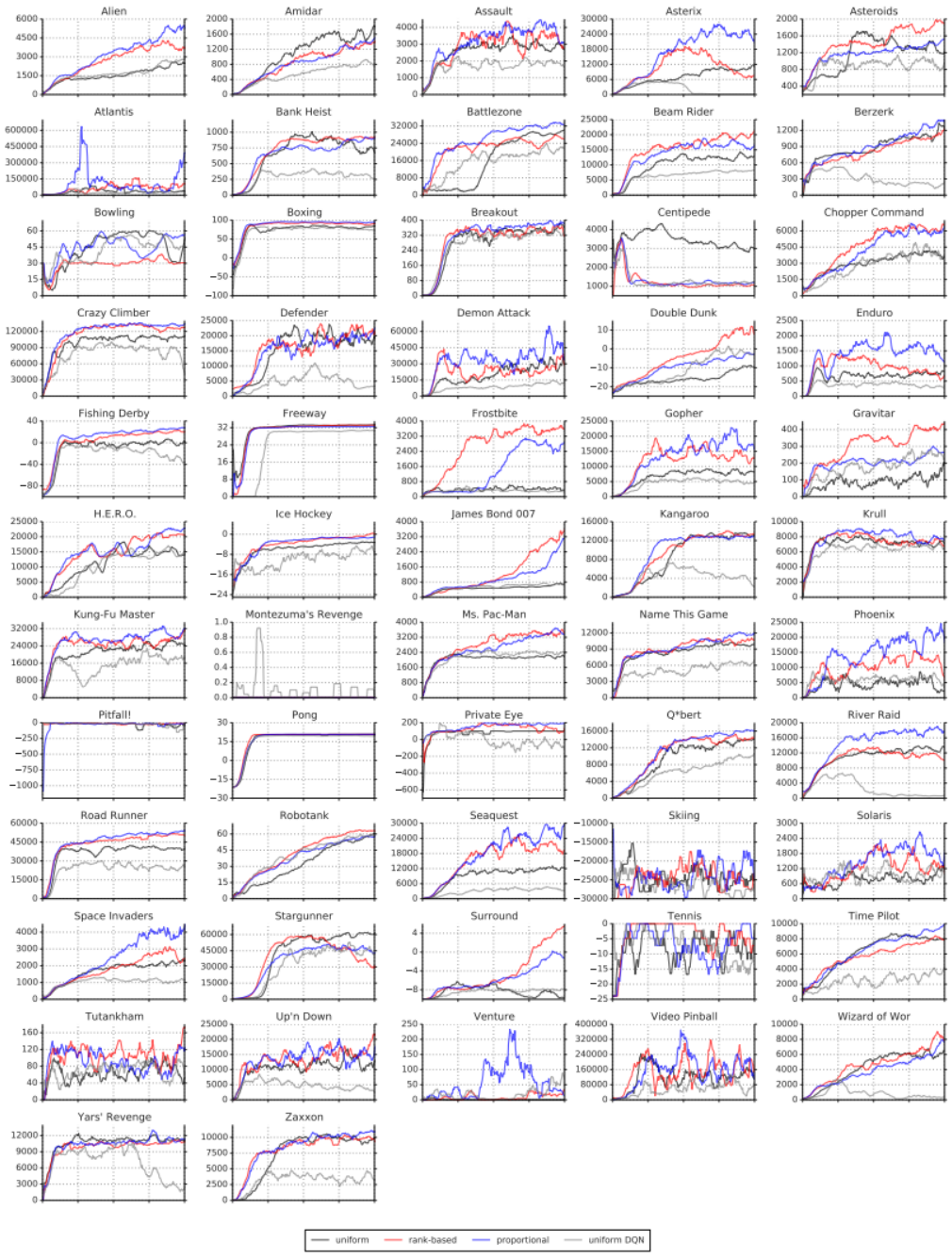


图 0.27　在雅达利基准测试套件所有 57 款游戏上，Double DQN（黑色为均匀基准）、基于排名的优先级回放（红色）、成比例的优先级级排序（蓝色）的学习曲线（原始得分）。每条曲线对应一次使用 test 评估（详见第 B.2.3 节）的训练过程，该过程包含 2 亿帧独立的数据帧，并采用 10 点移动平均平滑处理。原始 DQN 算法的学习曲线以灰色显示。更多子集学习曲线的详细展示请参见图 8。

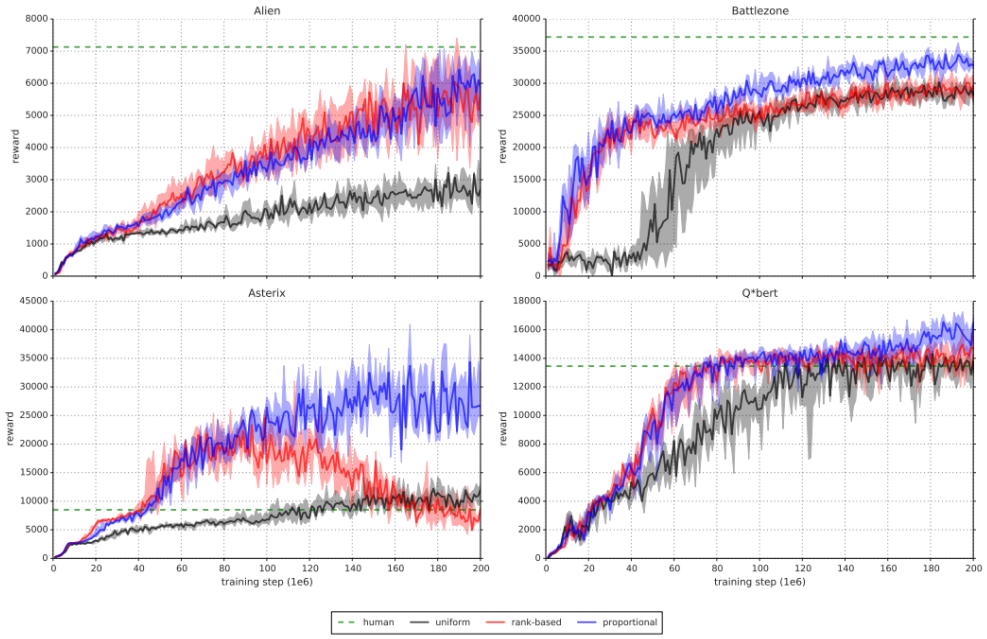


图 0.28　在选定的几款游戏中，基于排名的（红色）和成比例的（蓝色）优先级排序方法，与均匀采样的 Double DQN 基准模型（黑色）的详细学习曲线对比。实线代表得分中位数，阴影区域表示 8 次随机初始化实验结果的四分位距。绿色虚线代表人类玩家得分。尽管不同实验运行之间的波动性较大，但最终得分存在显著差异，学习速度也呈现明显区别。

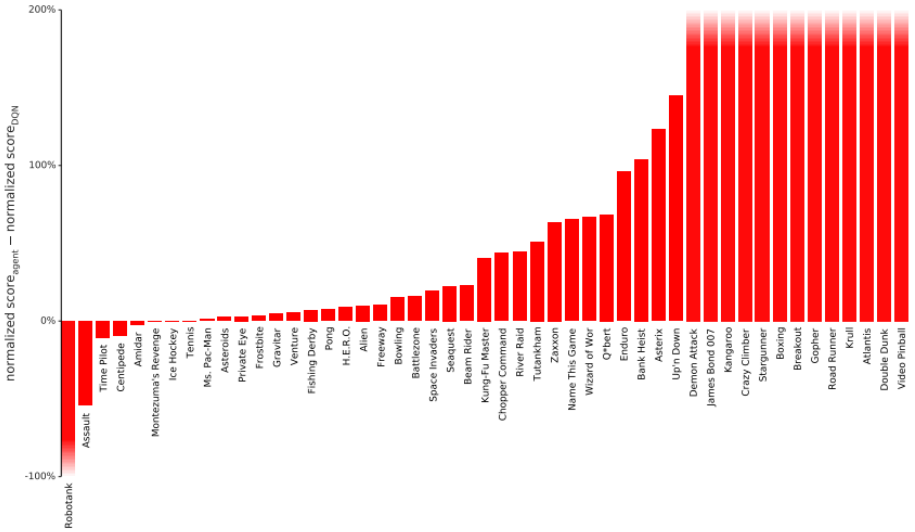


图 0.29　在 49 款采用“人类起始”方式的游戏上，带和不带基于排名的优先级回放的 DQN 之间标准化得分的差异（随机得分与人类得分之间的差距为 100%），显示在许多游戏中都有大幅提升。具体得分见表格 6。另见图 3，其中以 Double DQN 为基准。

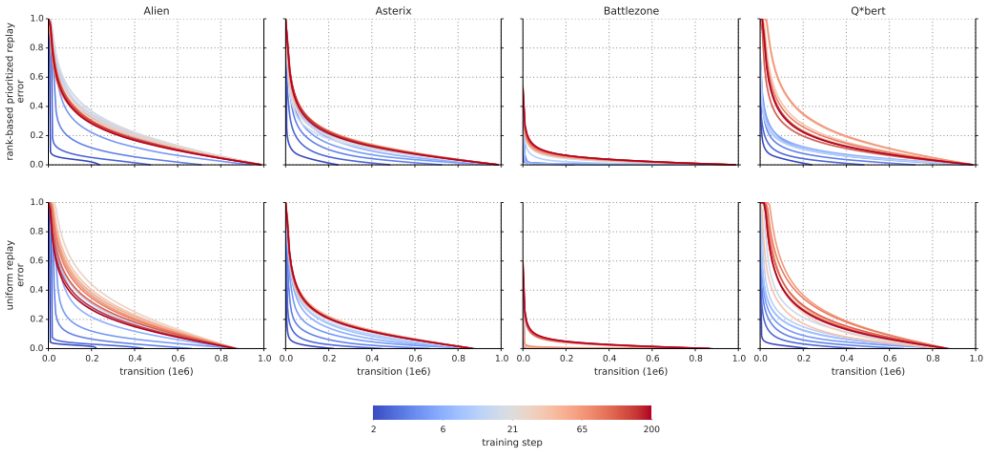


图 0.30　对部分雅达利游戏，回放记忆池中所有转移的上次观测到的绝对 TD 误差的可视化，这些误差已排序。线条按学习过程中的时间进行颜色编码，分辨率为 10^6 帧，训练初期用最冷的颜色，训练末期用最暖的颜色。我们观察到，在某些游戏中，误差一开始相当集中，但很快就变得分散，大致遵循重尾分布。这种现象在基于排名的优先级回放（上图）和均匀回放（下图）中都会出现，但优先级回放中该过程更快。

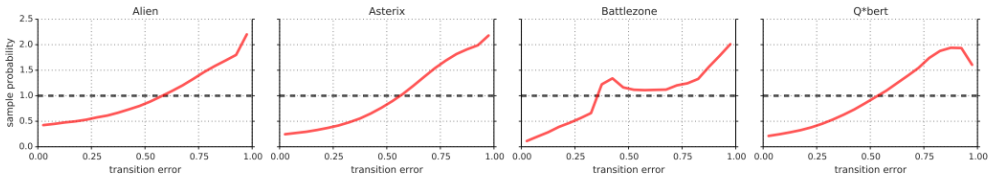


图 0.31　在训练初期，基于排名的优先级回放变体的**有效回放概率**随绝对 TD 误差的变化情况。与均匀基准（水平虚线）相比，该图展示了公式 (1) 在 $\alpha = 0.7$ 时的实际效果。效果虽不规则，但对于所选游戏，在定性上是相似的。

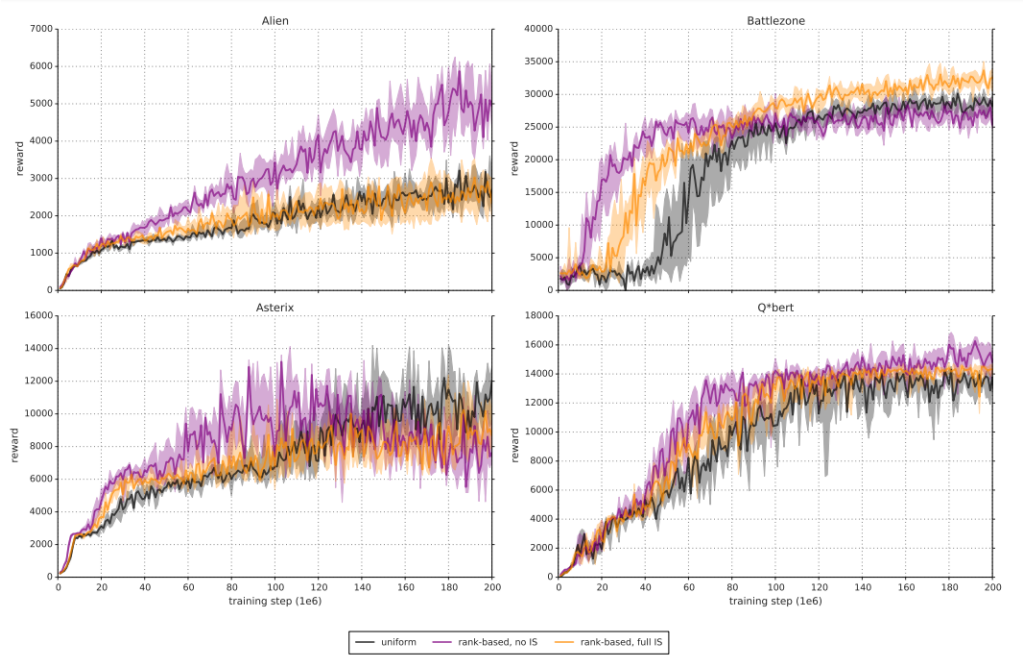


图 0.32　**重要性采样的效果：**这些学习曲线（如在图 8 中）展示了在几款选定的游戏上，基于排名的优先级排序如何受到完整重要性采样校正（即 $\beta = 1$，橙色）的影响，与均匀基准（黑色，$\alpha = 0$）和纯粹的、未校正的优先级回放（紫色，$\beta = 0$）相比。阴影区域对应四分位距。完全重要性采样校正的步长与均匀回放的步长相同。对于未校正的优先回放，步长缩小为原来的四分之一。与未校正的优先回放相比，重要性采样使学习的“激进程度”降低，一方面导致初始学习速度变慢，但另一方面降低了过早收敛的风险，有时还能带来更好的最终结果。与均匀回放相比，完全校正的优先级排序平均表现更优。

	DQN		Double DQN (tuned)		
Game	baseline	rank-based	baseline	rank-based	proportional
Alien	7%	17%	14%	**19%**	12%
Amidar	8%	6%	10%	8%	**14%**
Assault	685%	631%	1276%	1381%	**1641%**
Asterix	-1%	123%	226%	303%	**431%**
Asteroids	-0%	2%	1%	**2%**	2%
Atlantis	478%	1480%	2335%	2419%	**4425%**
Bank Heist	25%	129%	**139%**	137%	128%
Battlezone	48%	63%	72%	75%	**87%**
Beam Rider	57%	80%	117%	**210%**	176%
Berzerk		22%	40%	33%	**47%**
Bowling	5%	20%	**31%**	15%	27%
Boxing	246%	641%	**676%**	665%	632%
Breakout	1149%	**1823%**	1397%	1298%	1407%
Centipede	22%	12%	**23%**	19%	18%
Chopper Command	29%	**73%**	34%	48%	72%
Crazy Climber	179%	429%	448%	507%	**522%**
Defender		151%	**207%**	176%	155%
Demon Attack	390%	596%	2152%	1888%	**2256%**
Double Dunk	-350%	669%	981%	**2000%**	1169%
Enduro	68%	164%	158%	233%	**239%**
Fishing Derby	91%	98%	98%	**106%**	105%
Freeway	101%	111%	113%	**113%**	109%
Frostbite	2%	5%	33%	**83%**	69%
Gopher	120%	836%	728%	1679%	**2792%**
Gravitar	-1%	**4%**	-2%	1%	-1%
H.E.R.O.	47%	56%	55%	**80%**	78%
Ice Hockey	58%	58%	71%	**93%**	85%
James Bond 007	94%	311%	161%	**1172%**	1038%
Kangaroo	98%	339%	421%	**458%**	384%
Krull	283%	**1051%**	590%	598%	653%
Kung-Fu Master	56%	97%	146%	**153%**	151%
Montezuma's Revenge	1%	0%	0%	**1%**	-0%
Ms. Pac-Man	4%	5%	7%	**11%**	11%
Name This Game	73%	138%	143%	173%	**200%**
Phoenix		270%	202%	284%	**474%**
Pitfall!		2%	3%	-1%	**5%**
Pong	102%	110%	**111%**	110%	110%
Private Eye	-1%	2%	-2%	0%	-1%
Q*bert	37%	**106%**	91%	82%	93%
River Raid	25%	70%	74%	81%	**128%**
Road Runner	136%	**854%**	643%	780%	850%
Robotank	863%	752%	**872%**	828%	815%
Seaquest	6%	29%	36%	63%	**97%**
Skiing		-122%	33%	**44%**	38%
Solaris		-21%	-14%	**3%**	2%
Space Invaders	99%	118%	191%	291%	**693%**
Stargunner	378%	660%	653%	**689%**	580%
Surround		29%	77%	**103%**	58%
Tennis	130%	130%	93%	110%	**132%**
Time Pilot	100%	89%	140%	113%	**176%**
Tutankham	16%	**67%**	63%	35%	17%
Up'n Down	28%	173%	200%	125%	**313%**
Venture	4%	9%	0%	7%	**22%**
Video Pinball	-5%	4042%	7221%	5727%	**7367%**
Wizard of Wor	-15%	52%	144%	131%	**177%**
Yars' Revenge		**11%**	10%	7%	10%
Zaxxon	4%	68%	102%	113%	**113%**

表 15　57 款雅达利游戏的标准化得分（随机得分为 0%，人类得分为 100%），每个游戏均采用单次训练运行，使用人类起始评估方法（见第 B.2.3 节）。基准测试数据源自[23]，标准化得分的计算方式见公式 (4)。

			DQN			Double DQN (tuned)		
Game	random	human	baseline	Gorila	rank-b	baseline	rank-b.	prop.
Alien	128.3	6371.3	570.2	813.5	1191.0	1033.4	**1334.7**	900.5
Amidar	11.8	1540.4	133.4	189.2	98.9	169.1	129.1	**218.4**
Assault	166.9	628.9	3332.3	1195.8	3081.3	6060.8	6548.9	**7748.5**
Asterix	164.5	7536.0	124.5	3324.7	9199.5	16837.0	22484.5	**31907.5**
Asteroids	871.3	36517.3	697.1	953.6	1677.2	1193.2	**1745.1**	1654.0
Atlantis	13463.0	26575.0	76108.0	**629166.5**	207526.0	319688.0	330647.0	593642.0
Bank Heist	21.7	644.5	176.3	399.4	823.7	**886.0**	876.6	816.8
Battlezone	3560.0	33030.0	17560.0	19938.0	22250.0	24740.0	25520.0	**29100.0**
Beam Rider	254.6	14961.0	8672.4	3822.1	12041.9	17417.2	**31181.3**	26172.7
Berzerk	196.1	2237.5			644.0	1011.1	865.9	**1165.6**
Bowling	35.2	146.5	41.2	54.0	58.0	**69.6**	52.0	65.8
Boxing	-1.5	9.6	25.8	**74.2**	69.6	73.5	72.3	68.6
Breakout	1.6	27.9	303.9	313.0	**481.1**	368.9	343.0	371.6
Centipede	1925.5	10321.9	3773.1	**6296.9**	2959.4	3853.5	3489.1	3421.9
Chopper Command	644.0	8930.0	3046.0	3191.8	**6685.0**	3495.0	4635.0	6604.0
Crazy Climber	9337.0	32667.0	50992.0	65451.0	109337.0	113782.0	127512.0	**131086.0**
Defender	1965.5	14296.0			20634.0	**27510.0**	23666.5	21093.5
Demon Attack	208.3	3442.8	12835.2	14880.1	19478.8	69803.4	61277.5	**73185.8**
Double Dunk	-16.0	-14.4	-21.6	-11.3	-5.3	-0.3	**16.0**	2.7
Enduro	-81.8	740.2	475.6	71.0	1265.6	1216.6	1831.0	**1884.4**
Fishing Derby	-77.1	5.1	-2.3	4.6	3.5	3.2	**9.8**	9.2
Freeway	0.1	25.6	25.8	10.2	28.4	28.8	**28.9**	27.9
Frostbite	66.4	4202.8	157.4	426.6	288.7	1448.1	**3510.0**	2930.2
Gopher	250.0	2311.0	2731.8	4373.0	17478.2	15253.0	34858.8	**57783.8**
Gravitar	245.5	3116.0	216.5	**538.4**	351.0	200.5	269.5	218.0
H.E.R.O.	1580.3	25389.4	12952.5	8963.4	15150.9	14892.5	**20889.9**	20506.4
Ice Hockey	-9.7	0.5	-3.8	-1.7	-3.8	-2.5	**-0.2**	-1.0
James Bond 007	33.5	368.5	348.5	444.0	1074.5	573.0	**3961.0**	3511.5
Kangaroo	100.0	2739.0	2696.0	1431.0	9053.0	11204.0	**12185.0**	10241.0
Krull	1151.9	2109.1	3864.0	6363.1	**11209.5**	6796.1	6872.8	7406.5
Kung-Fu Master	304.0	20786.8	11875.0	20620.0	20181.0	30207.0	**31676.0**	31244.0
Montezuma's Revenge	25.0	4182.0	50.0	**84.0**	44.0	42.0	51.0	13.0
Ms. Pac-Man	197.8	15375.0	763.5	1263.0	964.7	1241.3	**1865.9**	1824.6
Name This Game	1747.8	6796.0	5439.9	9238.5	8738.5	8960.3	10497.6	**11836.1**
Phoenix	1134.4	6686.2			16107.8	12366.5	16903.6	**27430.1**
Pitfall!	-348.8	5989.8			-193.7	-186.7	-427.0	-14.8
Pong	-18.0	15.5	16.2	16.7	18.7	**19.1**	18.9	18.9
Private Eye	662.8	64169.1	298.2	**2598.6**	2202.3	-575.5	670.7	179.0
Q*bert	183.0	12085.0	4589.8	7089.8	**12740.5**	11020.8	9944.0	11277.0
River Raid	588.3	14382.2	4065.3	5310.3	10205.5	10838.4	11807.2	**18184.4**
Road Runner	200.0	6878.0	9264.0	43079.8	**57207.0**	43156.0	52264.0	56990.0
Robotank	2.4	8.9	58.5	**61.8**	51.3	59.1	56.2	55.4
Seaquest	215.5	40425.8	2793.9	10145.9	11848.8	14498.0	25463.7	**39096.7**
Skiing	-15287.4	-3686.6			-29404.3	-11490.4	**-10169.1**	-10852.8
Solaris	2047.2	11032.6			134.6	810.0	**2272.8**	2238.2
Space Invaders	182.6	1464.9	1449.7	1183.3	1696.9	2628.7	3912.1	**9063.0**
Stargunner	697.0	9528.0	34081.0	14919.2	58946.0	58365.0	**61582.0**	51959.0
Surround	-9.7	5.4			-5.3	1.9	**5.9**	-0.9
Tennis	-21.4	-6.7	-2.3	-0.7	-2.3	-7.8	-5.3	-2.0
Time Pilot	3273.0	5650.0	5640.0	**8267.8**	5391.0	6608.0	5963.0	7448.0
Tutankham	12.7	138.3	32.4	**118.5**	96.5	92.2	56.9	33.6
Up 'n' Down	707.2	9896.1	3311.3	8747.7	16626.5	19086.9	12157.4	**29443.7**
Venture	18.0	1039.0	54.0	**523.4**	110.0	21.0	94.0	244.0
Video Pinball	20452.0	15641.1	20228.1	112093.4	214925.3	367823.7	295972.8	**374886.9**
Wizard of Wor	804.0	4556.0	246.0	**10431.0**	2755.0	6201.0	5727.0	7451.0
Yars' Revenge	1476.9	47135.2			**6626.7**	6270.6	4687.4	5965.1
Zaxxon	475.0	8443.0	831.0	6159.4	5901.0	8593.0	9474.0	**9501.0**

表 16　在原始的 49 款雅达利游戏以及另外 8 款可评估的游戏上，采用“人类起始”方式进行评估得到的原始分数。人类、随机、DQN 以及优化版的 Double DQN 分数来自[23]。需要注意的是，[21] 中的研究结果使用了更大数据量和计算资源，但在这方面，其他方法相互之间更具直接可比性。

彩虹：深度强化学习改进方法的结合[1,2]

摘要

深度强化学习社区已经对 DQN 算法提出了若干独立的改进。然而，目前尚不清楚这些扩展之间是否互为补充，以及能否有效地结合起来。本文考察了 DQN 算法的六种扩展，并通过实证研究探讨它们的组合效果。实验结果表明，这些扩展的结合在 Atari 2600 基准测试中达到了最先进的性能，无论是数据效率还是最终表现方面。我们还提供了一项详细的消融实验结果，展示了每个组成部分对整体性能的贡献。

1 引言

近年来，将强化学习（RL）扩展到复杂序列决策问题的诸多成功，始于深度 Q 网络算法（DQN；Mnih 等人，2013，2015）。DQN 将 Q-learning 与卷积神经网络和经验回放相结合，使其能够仅从原始像素中学习，在许多 Atari 游戏上达到接近人类水平的表现。自那以后，研究者提出了许多扩展方法来提升其速度或稳定性。

双重 DQN（DDQN；van Hasselt, Guez, 和 Silver，2016）通过将引导动作的选择与评估解耦，解决了 Q-learning 的高估偏差问题（van Hasselt，2010）。优先经验回放（Schaul 等人，2015）通过更频繁地回放那些更具学习价值的转移，提高了数据效率。对决网络结构（Wang 等人，2016）通过分别表示状态值与动作优势，有助于在动作之间泛化。利用多步引导目标进行学习（Sutton，1988；Sutton 和 Barto，1998），如在 A3C（Mnih 等人，2016）中所使用的，改变了偏差-方差权衡，并帮助更快地将新观察到的奖励传播至更早访问的状态。分布的 Q-learning（Bellemare, Dabney, 和 Munos，2017）学习折扣回报的分类分布，而不是估计其均值。Noisy DQN（Fortunato 等人，2017）通过在网络层中引入随机性来进行探索。显然，这份列表远非穷尽。

这些算法在单独使用时都能显著提升性能。由于它们针对的是完全不同的问题，同时又基于一个共享的框架，因此它们很可能是可以结合的。在某些情况下，这种结合已经出现：例如优先 DDQN 和对决 DDQN 都使用了双重 Q-learning，并

[1]原文：Hessel M, Modayil J, Van Hasselt H, et al. Rainbow: Combining improvements in deep reinforcement learning[C]//Proceedings of the AAAI conference on artificial intelligence. 2018, 32(1).

[2]译者：李庆生。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

且对决 DDQN 也与优先回放相结合。本文提出研究一种结合上述所有方法的智能体。我们展示了如何将这些不同的思想整合起来，并证明它们确实是互补的。事实上，它们的结合在 Arcade Learning Environment（Bellemare 等人，2013）的 57 款 Atari 2600 游戏基准测试中，达到了新的最先进性能，无论是在数据效率还是最终表现方面。最后，我们还通过消融实验结果展示了各组成部分的独立贡献。

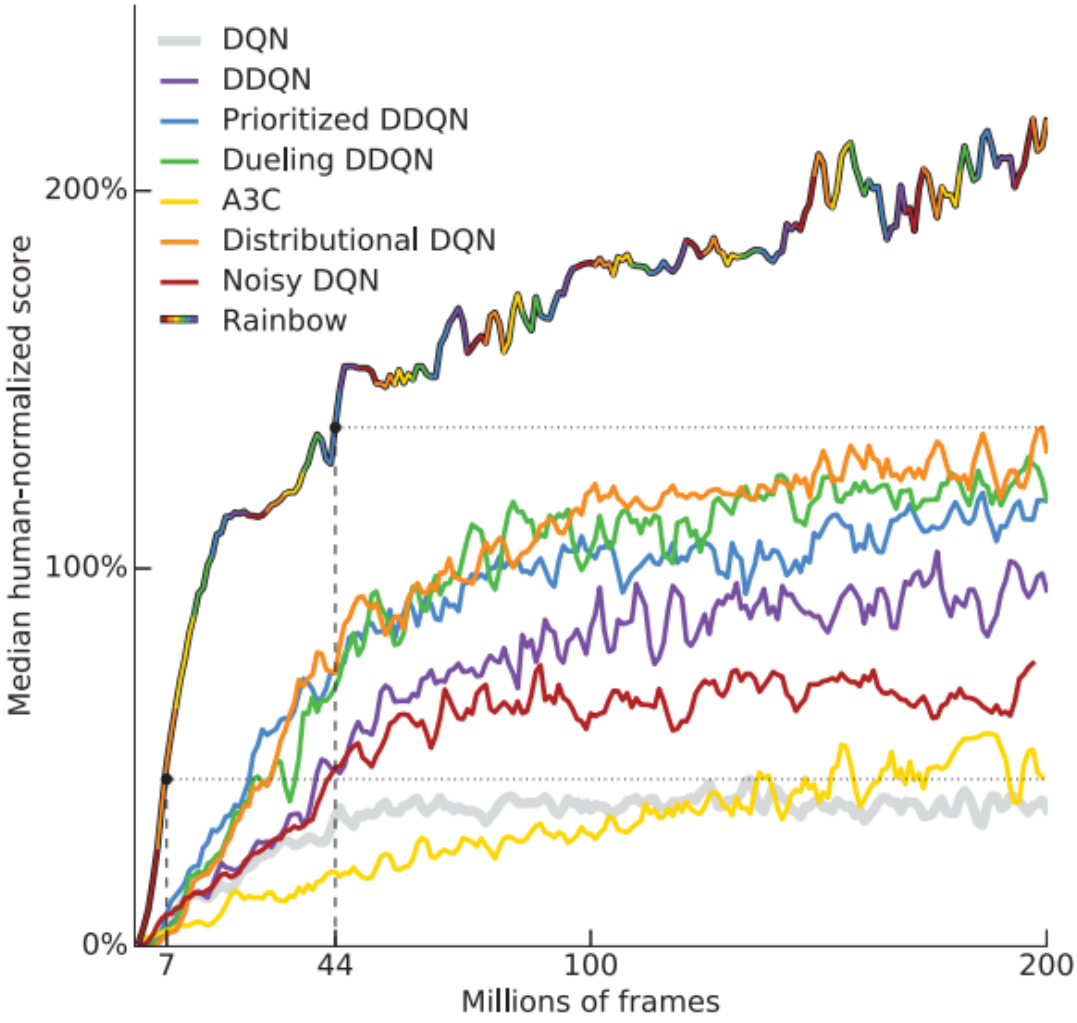


图 0.33　对 57 款雅达利游戏进行中位数人机归一化性能测试。将彩虹（Rainbow）与 DQN 算法及六种已发表基准模型进行对比。实验结果显示，彩虹在 700 万帧后达到 DQN 最佳性能，4400 万帧时彩虹超越所有基准模型，最终性能实现显著提升。曲线采用 5 点移动平均法进行平滑处理。

2 背景

强化学习所要解决的问题是：智能体在环境中学习如何行动，以最大化一个标量奖励信号。智能体不会得到直接的监督，例如，它不会被告知哪一个动作才

是最优的。

智能体与环境

在每一个离散时间步 $t = 0, 1, 2, \ldots$，环境向智能体提供一个观测 S_t，智能体则通过选择一个动作 A_t 来作出响应。随后，环境提供下一个奖励 R_{t+1}、折扣因子 γ_{t+1}，以及下一个状态 S_{t+1}。这一交互过程被形式化为一个马尔可夫决策过程（Markov Decision Process, MDP），记为元组 $\langle S, A, T, r, \gamma \rangle$，其中 S 是有限状态集合，A 是有限动作集合，$T(s, a, s') = P[S_{t+1} = s' \mid S_t = s, A_t = a]$ 是（随机的）转移函数，$r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$ 是奖励函数，$\gamma \in [0, 1]$ 是折扣因子。在我们的实验中，MDPs 具有恒定的 $\gamma_t = \gamma$，除非在终止时 $\gamma_t = 0$，但算法是用一般形式表达的。

在智能体方面，动作选择由一个策略 π 来定义，它是关于每个状态下动作的概率分布。从时间 t 遇到的状态 S_t 出发，我们定义折扣回报

$$G_t = \sum_{k=0}^{\infty} \gamma_t^{(k)} R_{t+k+1},$$

其中 G_t 是智能体在未来收集到的折扣奖励总和。折扣因子 $\gamma_t^{(k)}$ 表示 k 步之后的奖励的折扣，其定义为前 k 个折扣因子的连乘：

$$\gamma_t^{(k)} = \prod_{i=1}^{k} \gamma_{t+i}.$$

智能体的目标是通过找到一个良好的策略，最大化期望折扣回报。

策略可能通过直接学习获得，也可能通过其他学习量构建而来。在基于价值的强化学习中，智能体学习的是期望折扣回报的一个估计量，称为 *价值函数*。当遵循策略 π 并从某一状态 s 开始时，状态值函数为

$$v^{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid S_t = s],$$

状态-动作值函数为

$$q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a].$$

一种常见的从状态-动作值函数导出新策略的方法是 ϵ-贪婪方法：以概率 $1 - \epsilon$ 选择使 $q(s, a)$ 最大的动作（称为 *贪婪动作*），以概率 ϵ 随机选择动作。这样的策略用于引入一种 *探索*：通过随机选择与当前估计不一致的次优动作，智能体能够发现并在适当的时候纠正自身估计中的错误。主要的限制是很难发现可延伸到遥远未来的其他行动方针，这促使人们进行更有针对性的探索形式的研究。

深度强化学习与 DQN

在状态空间和/或动作空间很大的情况下，想要为每一个状态-动作对单独学习 Q 值估计是不可行的。在深度强化学习中，我们使用深度（即多层）神经网络来表示智能体的各种组成部分，例如策略 $\pi(s,a)$ 或值函数 $q(s,a)$。这些网络的参数通过梯度下降来训练，以最小化某个合适的损失函数。

在 DQN（Mnih 等人，2015）中，深度网络与强化学习成功地结合在一起，使用卷积神经网络来逼近给定状态 S_t 的动作值函数（输入为一堆原始像素帧）。在每一个步骤中，基于当前状态，智能体以 ϵ-贪婪方式选择一个动作 A_t 并执行，然后将转移 $(S_t, A_t, R_{t+1}, \gamma_{t+1}, S_{t+1})$ 存入经验回放缓冲区（Lin, 1992），该缓冲区最多保存最近的一百万条转移。

神经网络的参数通过随机梯度下降进行优化，以最小化以下损失函数：

$$\left(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_{\theta}(S_t, A_t)\right)^2, \tag{1}$$

其中 t 是从经验回放中随机抽取的一个时间步。损失的梯度被反向传播到主网络的参数 θ 中（该网络也用于选择动作）；而 $\bar{\theta}$ 表示目标网络的参数，这是在线网络的一个周期性拷贝，并不会直接优化。优化采用 RMSProp 算法（Tieleman 和 Hinton，2012），它是一种随机梯度下降的变体，在经验回放中均匀采样的小批量上执行。这意味着在上述损失中，时间索引 t 将是从最近的一百万条转移中均匀抽取的一个随机索引，而不是当前时间步。经验回放与目标网络的结合，使得 Q 值的学习相对稳定，并在若干 Atari 游戏上实现了超越人类水平的表现。

3 DQN 的扩展

DQN 是一个重要的里程碑，但现在已经发现该算法存在若干局限性，因此提出了许多扩展方法。我们在此选择六种扩展，每一种都针对某一局限性并提升了整体性能。为了使选择的范围可控，我们挑选了一组解决不同问题的扩展方法（例如，在众多与探索相关的方法中只选取其中一种）。

双重 Q-learning.

传统的 Q-learning 会受到高估偏差的影响，这是由于式 (1) 中的最大化步骤所导致的，这可能会损害学习效果。双重 Q-learning（van Hasselt, 2010）通过在引导目标的最大化步骤中，将动作的选择与其评估解耦，从而解决了这一高估问题。该方法可以有效地与 DQN 相结合（van Hasselt, Guez, and Silver, 2016），其损失函数

为：

$$\left(R_{t+1}+\gamma_{t+1}q_{\bar{\theta}}(S_{t+1},\arg\max_a q_\theta(S_{t+1},a))-q_\theta(S_t,A_t)\right)^2.$$

实验证明，这一改动能够减少 DQN 中存在的有害高估，从而提高性能。

优先经验回放.

DQN 从回放缓冲区中均匀采样。然而，理想情况下，我们希望更频繁地采样那些包含更多学习信息的转移。作为学习潜力的近似度量，优先经验回放（Schaul et al., 2015）以概率 p_t 来采样转移，该概率与最近一次出现的绝对 TD 误差成比例：

$$p_t \propto \left|R_{t+1}+\gamma_{t+1}\max_a q_\theta(S_{t+1},a)-q_\theta(S_t,A_t)\right|^\omega,$$

其中 ω 是一个超参数，用于控制分布的形状。新的转移以最大优先级被插入回放缓冲区，从而对最近的转移产生偏向。需要注意的是，即使某些转移几乎没有新的信息可供学习，但由于其随机性，它们也可能被偏向采样。

对决网络.

对决网络是一种为基于价值的强化学习设计的神经网络结构。它包含两个计算流：价值流（value stream）和优势流（advantage stream）。二者共享一个卷积编码器，并通过一个特殊的聚合器进行合并（Wang et al., 2016）。其对应的动作价值分解如下：

$$q_\theta(s,a)=v_\eta(f_\xi(s))+a_\psi(f_\xi(s),a)-\frac{1}{N_{\text{actions}}}\sum_{a'}a_\psi(f_\xi(s),a'),$$

其中，ξ,η,ψ 分别表示共享编码器 f_ξ、价值流 v_η 和优势流 a_ψ 的参数；而 $\theta=\{\xi,\eta,\psi\}$ 表示它们的整体参数集合。

对决网络的分解意义

在标准的 DQN 中，网络直接输出每个动作的 $Q(s,a)$ 值。然而，在许多状态下，动作之间的相对优劣并不重要。例如在游戏的终局状态，无论采取哪个动作结果都相同。此时直接学习 $Q(s,a)$ 往往效率较低。

为了解决这一问题，对决网络提出将 $Q(s,a)$ 分解为两部分：

- **状态价值**：$v_\eta(f_\xi(s))$，表示状态 s 的整体好坏；
- **动作优势**：$a_\psi(f_\xi(s),a)$，表示在状态 s 下，动作 a 相对于平均动作的优势程度。

其形式化表达为：

$$q_\theta(s,a) = v_\eta(f_\xi(s)) + a_\psi(f_\xi(s),a) - \frac{1}{N_{\text{actions}}}\sum_{a'} a_\psi(f_\xi(s),a').$$

为什么要减去平均值？

在分解

$$Q(s,a) = V(s) + A(s,a)$$

时，这种分解并不是唯一的。例如，任意常数 c 都可以生成新的等价分解：

$$Q(s,a) = (V(s)+c) + (A(s,a)-c).$$

为了避免这种不唯一性导致的训练不稳定，对决网络约束优势函数的均值为零：

$$\sum_a A(s,a) = 0,$$

通过减去平均值来固定分解，使得学习过程更加稳定。

意义总结

- **稳定性**：避免 $V(s)$ 和 $A(s,a)$ 之间相互“抢功劳”。
- **泛化性**：提升样本效率，加快收敛速度。

结构示意图

[node distance=1.2cm, every node/.style=font=, align=center, >=stealth, thick] [draw, rounded corners, minimum width=3.2cm, minimum height=0.9cm, fill=blue!10] (encoder) 共享卷积编码器 $f_\xi(s)$;

[draw, rounded corners, below left=1.2cm and 0.8cm of encoder, fill=green!10, minimum width=2.8cm, minimum height=0.9cm] (value) Value Stream v_η; [draw, rounded corners, below right=1.2cm and 0.8cm of encoder, fill=orange!10, minimum width=3.2cm, minimum height=0.9cm] (advantage) Advantage Stream a_ψ;

[draw, rounded corners, below=2.5cm of encoder, fill=purple!10, minimum width=4.5cm, minimum height=0.9cm] (agg) Aggregator: $Q(s,a) = V(s) + A(s,a) - \text{mean}(A)$;

[->, thick] (encoder.south) |- (value.north); [->, thick] (encoder.south) |- (advantage.north); [->, thick] (value.south) – ([yshift=-0.2cm]value.south) -| (agg.north west); [->, thick] (advantage.south) – ([yshift=-0.2cm]advantage.south) -| (agg.north east);

[above=0.2cm of encoder] 输入状态 s; [below=0.2cm of agg] 输出 Q 值 $Q(s,a)$;

多步学习.

在传统的 Q 学习中，只累积一个奖励，并在下一步使用贪婪动作进行自举。另一种方法是使用前向视角的多步目标（Sutton, 1988）。从给定状态 S_t 出发，截断的 n 步回报定义为：

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}. \tag{2}$$

于是，DQN 的多步变体通过最小化如下的替代损失函数来定义：

$$\left(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_{\bar{\theta}}(S_{t+n}, a') - q_\theta(S_t, A_t)\right)^2.$$

当 n 选取合适时，多步目标通常可以带来更快的学习效果（Sutton and Barto, 1998）。

多步学习的意义

在标准 Q 学习中，目标值是由单步奖励和下一步状态的估计值构成的：

$$Q(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a').$$

这种方法只利用了当前的一步奖励，并立即进行 bootstrap 更新，可能导致学习过程收敛较慢。多步学习使用截断的 n 步回报作为目标值。定义从状态 S_t 出发的 n 步回报为：

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma^k r_{t+k+1}.$$

其对应的更新目标为：

$$Q(s_t, a_t) \leftarrow R_t^{(n)} + \gamma^n \max_{a'} Q(s_{t+n}, a').$$

意义分析

- **偏差-方差权衡：**单步目标 ($n = 1$) 偏差较大，但方差较小；大步长 (n 较大) 方差较大，但偏差减小。通过选择合适的 n，可以在偏差与方差之间取得平衡。
- **奖励传播更快：**新获得的奖励能够更快地回传至之前的状态，而不是依赖逐步传播。这通常加速了学习过程。
- **更稳定的学习：**相较于完全依赖单步自举，多步回报利用了更多的真实奖励信号，从而降低了对不稳定 Q 估计的依赖。

分布的强化学习.

我们可以学习近似回报的分布，而非期望回报。最近，Bellemare、Dabney 和 Munos (2017) 提出了在离散支撑集 z 上放置概率质量来建模这种分布，其中 z 是一个包含 $N_{\text{atoms}} \in \mathbb{N}^+$ 个原子的向量，定义为

$$z^i = v_{\min} + (i-1)\frac{v_{\max} - v_{\min}}{N_{\text{atoms}} - 1}, \quad i \in \{1, \ldots, N_{\text{atoms}}\}.$$

在时刻 t，近似分布 d_t 定义在该支撑集上，每个原子 i 具有概率 $p_\theta^i(S_t, A_t)$，使得 $d_t = (z, p_\theta(S_t, A_t))$。学习的目标是更新参数 θ，使得该分布能够尽可能贴近真实的回报分布。

为了学习概率质量，一个关键的洞见是: 回报分布满足 Bellman 方程的一种变体。对于给定状态 S_t 和动作 A_t，最优策略 π^* 下的回报分布应当与一个目标分布相匹配。该目标分布是由下一状态 S_{t+1} 和最优动作 $a^*_{t+1} = \pi^*(S_{t+1})$ 的分布构造而成，通过折扣因子将其向零收缩，并通过奖励（或在随机情形下的奖励分布）进行平移。于是可以得到 Q-learning 的分布式变体：首先为目标分布构造一个新的支撑集，然后最小化分布 d_t 与目标分布 $d'_t \equiv (R_{t+1} + \gamma_{t+1} z,\ p_{\bar{\theta}}(S_{t+1}, \bar{a}^*_{t+1}))$ 之间的 Kullback-Leibler 散度：

$$D_{\text{KL}}(\Phi_z d'_t \parallel d_t), \tag{3}$$

其中，Φ_z 表示将目标分布投影到固定支撑 z 上的 L_2 投影，$\bar{a}^*_{t+1} = \arg\max_a q_{\bar{\theta}}(S_{t+1}, a)$ 是关于平均动作价值 $q_{\bar{\theta}}(S_{t+1}, a) = z^\top p_\theta(S_{t+1}, a)$ 的贪婪动作。

与非分布的情形类似，我们可以使用参数的冻结副本 $\bar{\theta}$ 来构造目标分布。参数化的分布可以通过神经网络来表示，就像 DQN 一样，但输出维度为 $N_{\text{atoms}} \times N_{\text{actions}}$。在输出的每个动作维度上独立地应用 softmax，以确保每个动作对应的分布被正确归一化。

1. 为什么要研究分布而不是期望？

传统 Q-learning 或 DQN 学的是期望：$q^\pi(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$, 其中 G_t 是未来累积奖励。但期望会丢失很多信息。例如：如果 G_t 的分布是 $[0, 10]$，概率各一半；或者分布是总是 5；这两种情况的期望都是 5，但含义完全不同。第一个情况风险很大（要么很差，要么很好），第二个情况结果很稳定。因此，学习整个分布可以帮助智能体更好地理解不确定性。

2. 如何表示分布？

我们不可能精确表示一个连续分布，所以选择在固定的支撑 z 上近似（离散化）。例如，设 $z = [z^1, z^2, \ldots, z^{N_{\text{atoms}}}]$, 这些点是均匀排列的（从 $v_{\min}$ 到 $v_{\max}$）。网络输出每个点上的概率 $p_\theta^i(s,a), \quad i = 1, \ldots, N_{\text{atoms}}$, 并保证 $\sum_{i=1}^{N_{\text{atoms}}} p_\theta^i(s,a) = 1$.

于是回报分布近似为：$d(s,a) = (z, p_\theta(s,a))$. 这就是一个“离散概率分布”，例如：$z = [0,2,4,6], \quad p = [0.1, 0.2, 0.6, 0.1]$.

3. Bellman 更新如何作用在分布上？

在传统 Q-learning 里，目标是：$y = R_{t+1} + \gamma \max_a q_{\bar{\theta}}(S_{t+1}, a)$. 在分布的 RL 里，我们希望更新整个分布。下一步的分布是 $d_{t+1} = (z, p_{\bar{\theta}}(S_{t+1}, a^*)), \quad a^* = \arg\max_a z^\top p_{\bar{\theta}}(S_{t+1}, a)$. 如果下一步的分布有一个原子 z^i，那么经过折扣和平移后，它变成：$t^i = R_{t+1} + \gamma z^i$. 这就是回报分布的“平移与缩放”。

4. 为什么需要投影 Φ_z？

问题：t^i 往往不在原来的固定支撑 z 上。例如，原子是 $[0,2,4,6]$，结果算出来 $t^i = 3.4$，它“落在”2 和 4 之间。怎么办？

做法是：把 t^i 的概率质量分配到最接近的两个原子上。

$$l = \lfloor \tfrac{t^i - v_{\min}}{\Delta z} \rfloor, \quad u = \lceil \tfrac{t^i - v_{\min}}{\Delta z} \rceil.$$

然后：

$$m_l \mathrel{+}= p^i_{\text{next}} \cdot (u - b), \quad m_u \mathrel{+}= p^i_{\text{next}} \cdot (b - l),$$

其中 $b = (t^i - v_{\min})/\Delta z$。这样就把质量“投影”到了原来的支撑 z 上。

数值例子 设 $v_{\min} = 0$, $v_{\max} = 6$, $N_{\text{atoms}} = 4$, $z = [0,2,4,6]$。下一步分布全在 $z^2 = 2$ 上（概率 1）。奖励 $R_{t+1} = 1$, $\gamma = 0.9$。于是

$$t = 1 + 0.9 \cdot 2 = 2.8.$$

它在 $z^2 = 2$ 和 $z^3 = 4$ 之间。于是：

$$m_2 = (4 - 2.8)/2 = 0.6, \quad m_3 = (2.8 - 2)/2 = 0.4.$$

最终目标分布是 $[0, 0.6, 0.4, 0]$。

5. 为什么这样做更好？

- **更强的学习信号**：不是只学习一个数字，而是学习一整个概率分布，信息量更大。
- **不确定性**：分布能告诉我们风险大小（方差），而不仅是期望。
- **更稳定**：实验表明在 Atari 等任务中，其学习曲线更平滑、更快。

噪声网络.

在某些环境中（例如 *Montezuma's Revenge*），使用 ϵ-greedy 策略进行探索的局限性非常明显，因为智能体需要执行许多步动作才能获得第一个奖励。为了解决这一问题，噪声网络（Fortunato 等人，2017）提出了一种带噪声的线性层，该层结合了确定性部分与噪声部分，其形式为：

$$y = (b + Wx) + (b^{\text{noisy}} \odot \epsilon_b + (W^{\text{noisy}} \odot \epsilon_w)x), \tag{3}$$

其中 ϵ_b 和 ϵ_w 是随机变量，符号 $\odot$ 表示逐元素乘法。该变换可以直接替代标准的线性层

$$y = b + Wx.$$

随着训练的进行，网络能够学习在某些情况下忽略噪声部分，但在状态空间的不同区域中这种忽略的速度不同。这样，噪声网络允许实现一种依赖状态的探索，并具备类似自退火的特性。(自退火 = 网络通过训练自动学会逐渐减弱或保留探索噪声，而不是依赖人为设定的 ϵ。)

3 集成智能体

在本文中，我们将前面介绍的所有组件整合为一个单一的智能体，称为 **Rainbow**。首先，我们将一步的分布的损失（公式 (3)）替换为多步变体。我们通过以下方式构造目标分布：根据累积折扣对状态 S_{t+n} 的价值分布进行收缩，并用截断的 n 步折扣回报进行平移。这对应于定义目标分布为

$$d_t^{(n)} = \left(R_t^{(n)} + \gamma_t^{(n)} z,\ p_\theta(S_{t+n}, a^*_{t+n})\right).$$

由此得到的损失函数为

$$D_{\text{KL}}\left(\Phi_z d_t^{(n)} \,\|\, d_t\right),$$

其中 Φ_z 表示对支撑集 z 的投影。

我们将多步分布式损失与双重 Q-learning 结合起来：在状态 S_{t+n} 中，使用 *在线网络*所选择的贪婪动作 a^*_{t+n} 作为自举动作，而该动作的价值评估则通过 *目标网络*完成。

在标准的比例优先经验回放（Schaul et al. 2015）中，使用绝对 TD 误差对转移进行优先级排序。在分布的设定下，可以利用平均动作价值来计算这一指标。然而，在我们的实验中，所有分布式 Rainbow 变体均采用 KL 损失作为优先级指标，

因为这是算法实际最小化的对象：

$$p_t \propto \left(D_{\mathrm{KL}}\left(\Phi_z d_t^{(n)} \,\|\, d_t\right)\right)^{\omega}.$$

使用 KL 损失作为优先级可能在噪声较大的随机环境中更具鲁棒性，因为即使回报并非确定性的，损失依然可以继续减小。

网络采用一种对决网络结构，并针对回报分布进行了改造。网络首先通过共享表示 $f_\xi(s)$，然后将其输入到 价值流 v_η（具有 N_{atoms} 个输出），以及 优势流 a_ψ（具有 $N_{\text{atoms}} \times N_{\text{actions}}$ 个输出）。其中，$a_\psi^i(f_\xi(s), a)$ 表示对应于原子 i 和动作 a 的输出。

对于每个原子 z^i，价值流和优势流按照 dueling DQN 的方式进行聚合，然后通过 softmax 层得到归一化的参数化分布，用于估计回报分布：

$$p_\theta^i(s,a) = \frac{\exp\left(v_\eta^i(\phi) + a_\psi^i(\phi,a) - \bar{a}_\psi^i(s)\right)}{\sum_j \exp\left(v_\eta^j(\phi) + a_\psi^j(\phi,a) - \bar{a}_\psi^j(s)\right)},$$

其中 $\phi = f_\xi(s)$，且

$$\bar{a}_\psi^i(s) = \frac{1}{N_{\text{actions}}} \sum_{a'} a_\psi^i(\phi, a').$$

接下来，我们将所有的线性层替换为噪声线性层（见公式 (4)）。在这些噪声线性层中，我们采用了 分解高斯噪声（factorised Gaussian noise）（Fortunato et al. 2017），以减少独立噪声变量的数量。

分解高斯噪声（Factorised Gaussian noise）的主要思想是通过输入端噪声和输出端噪声的外积来构造权重噪声，从而减少独立随机变量的数量。具体而言，设

$$\epsilon^p \sim \mathcal{N}(0, I_n), \quad \epsilon^q \sim \mathcal{N}(0, I_m),$$

并定义非线性变换

$$f(x) = \mathrm{sgn}(x)\sqrt{|x|}.$$

则权重噪声和偏置噪声为

$$\epsilon_w = f(\epsilon^p) \cdot f(\epsilon^q)^\top, \quad \epsilon_b = f(\epsilon^p).$$

这种方法相比为每个权重采样独立噪声，显著减少了采样开销，同时仍保持较强的探索能力。

4 实验方法

本节介绍用于配置和评估学习智能体的方法和实验设置。

评估方法

我们在 57 个 Atari 2600 游戏上评估了所有智能体，这些游戏来自 Arcade Learning Environment (Bellemare et al. 2013). 我们遵循 Mnih et al. (2015) 和 van Hasselt et al. (2016) 的训练与评估流程。在训练过程中，每经过 100 万步环境交互，就暂停学习并评估最新智能体 50 万帧，计算智能体的平均得分。每个回合在 108K 帧（约 30 分钟的模拟游戏时间）时被截断，这与 van Hasselt et al. (2016) 一致。

为了比较不同智能体的表现，我们对每个游戏的得分进行归一化，其中随机智能体的得分对应 0%，而人类专家的平均得分对应 100%。归一化得分可以在所有 Atari 游戏上进行聚合，用于评估整体表现。通常采用所有游戏的中位人类归一化表现 作为指标。此外，我们还考虑智能体得分超过人类表现某一比例的游戏数量，以进一步分析中位数提升的来源。相比之下，均值人类归一化表现信息量可能更低，因为它常常被少数游戏（如 *Atlantis*）所主导——在这些游戏中，智能体得分远远高于人类。

除了在训练过程中跟踪随环境步数变化的中位表现外，在训练结束时，我们还对最佳的智能体快照进行两种不同的测试：

- **No-ops 启动**：在每个回合开始时插入一个随机数量（最多 30 个）的空操作动作（训练阶段也采用该方式）。
- **Human starts 启动**：从人类专家的轨迹初始部分中随机抽取状态作为回合的起点 (Nair et al. 2015)。

这两种测试方式的性能差异可以反映出智能体在多大程度上**过拟合于自身轨迹**。

超参数调优

Rainbow 的各个组件都有若干超参数。由于超参数的组合空间过大，无法进行穷举搜索，因此我们进行了有限的调优。对于每个组件，我们从引入该组件的论文中使用的数值作为初始值，并通过手动坐标下降方法调节其中最为敏感的超参数。

在前 200K 帧内，DQN 及其变体不会执行学习更新，以确保更新之间具有足够的不相关性。我们发现，在使用优先级经验回放时，可以更早地开始学习，仅需在 80K 帧之后即可。

DQN 的初始探索率 $\epsilon = 1$，对应于完全随机动作选择；然后在前 4M 帧中逐步退火至最终值 0.1（在后续变体中降低至 0.01）。当使用 Noisy Nets 时，我们采用完全贪婪策略（$\epsilon = 0$），并将噪声流中权重的初始化超参数设置为 $\sigma_0 = 0.5$。对于不使用 Noisy Nets 的智能体，我们采用 ϵ-greedy 策略，但探索率退火得比以往更快，

在前 250K 帧中退火至 0.01。

我们使用 Adam 优化器 (Kingma and Ba 2014)，其对学习率的敏感性小于 RMSProp。DQN 使用的学习率为 $\alpha = 0.00025$。在所有 Rainbow 变体中，我们使用了 $\alpha/4$ 的学习率（在集合 $\{\alpha/2, \alpha/4, \alpha/6\}$ 中选择），并将 Adam 的 ϵ 超参数设置为 1.5×10^{-4}。

多步学习中的 n 值是 Rainbow 的一个敏感超参数。我们比较了 $n = 1, 3, 5$ 的情况。结果表明，$n = 3$ 和 $n = 5$ 在初期都表现良好，但最终 $n = 3$ 的整体性能最佳。

在经验回放优先级中，我们使用推荐的比例型变体，优先级指数 ω 取 0.5，并在训练过程中将重要性采样指数 β 从 0.4 线性增加至 1。优先级指数 ω 在 $\{0.4, 0.5, 0.7\}$ 中进行调优。使用分布式 DQN 的 KL 损失作为优先级时，我们观察到性能对 ω 的选择非常鲁棒。

所有 57 个游戏的超参数（见 Table 1）完全相同，即 Rainbow 智能体实际上是一个在所有游戏中均表现良好的单一智能体设置。

表 17　Rainbow 超参数设置

参数	取值
开始学习的最小历史帧数	80K 帧
Adam 学习率	0.0000625
探索率 ϵ	0.0
Noisy Nets 初始 σ_0	0.5
目标网络更新周期	32K 帧
Adam ϵ	1.5×10^{-4}
优先级类型	比例型 (proportional)
优先级指数 ω	0.5
优先级重要性采样 β	$0.4 \rightarrow 1.0$
多步回报 n	3
分布原子数 (atoms)	51
分布最小/最大值	$[-10, 10]$

5 分析

在本节中，我们分析主要的实验结果。首先，我们展示 Rainbow 与若干已发表的智能体相比具有更优的表现。随后，我们进行了消融实验（ablation study），比较了多个 Rainbow 的变体，每个变体对应于移除 Rainbow 的某一个组成部分。

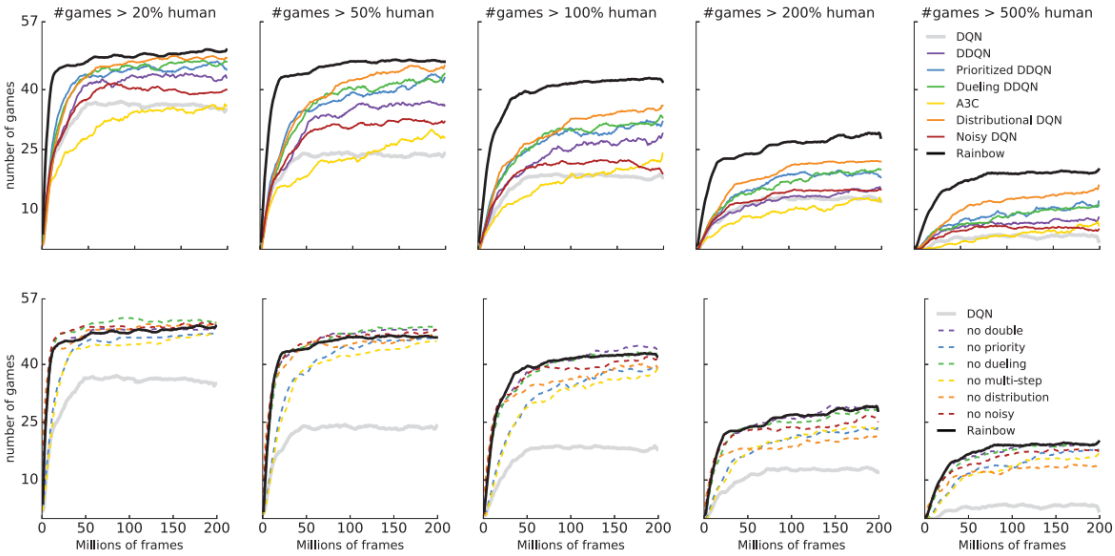


图 0.34　每张图表展示了多个智能体在不同时间点达成至少达到人类表现基准分数的游戏数量变化趋势。从左至右依次对应 20%、50%、100%、200% 和 500% 的性能阈值。首行对比显示，Rainbow 算法与基线模型的表现差异；次行则呈现该算法经过功能消融处理后的性能对比结果。

与已有基准的比较.

在图 1 中，我们将 Rainbow 的表现（通过所有游戏的中位人类归一化得分衡量）与 A3C、DQN、DDQN、Prioritized DDQN、Dueling DDQN、Distributional DQN 以及 Noisy DQN 的对应曲线进行了比较。我们感谢 Dueling 与 Prioritized 智能体作者提供了它们的学习曲线，同时我们报告了对 DQN、A3C、DDQN、Distributional DQN 和 Noisy DQN 的重新实验结果。可以看到，Rainbow 在数据效率与最终性能两个方面均显著优于所有基线。值得注意的是，Rainbow 在 700 万帧时即可达到 DQN 的最终性能，在 4400 万帧时超越了所有基线的最佳最终性能，并最终取得了显著提升的表现。

在训练结束后的最终评估中，Rainbow 在 no-ops 启动的测试模式下取得了 231% 的中位得分；在人类启动模式下取得了 153% 的中位得分。在表 2 中，我们将这些结果与各个基线智能体已发表的中位得分进行了比较。

在图 2 中，我们绘制了智能体在不同游戏中达到某一水平人类归一化性能的数量。从左到右，子图分别展示了不同智能体在多少个游戏中至少达到了 20%、50%、100%、200% 和 500% 的人类归一化性能。该结果有助于我们识别整体性能提升的来源。需要注意的是，在所有性能水平下，Rainbow 与其他智能体之间的差距都是明显的：Rainbow 智能体不仅在基线智能体已经表现良好的游戏上进一步提升了得分，同时也在基线智能体仍远低于人类表现的游戏上取得了显著改进。

表 18　彩虹算法与各基准方法的最佳智能体快照归一化得分中位数对比。带星号标注的方法数据源自其原始论文，其中 DQN 算法的分数来自《对抗网络》研究论文（因其未完整报告全部 57 局游戏数据），其余数据均来自我们自主研发的实现版本。

Agent	No-ops	Human starts
DQN	79%	68%
DDQN (*)	117%	110%
Prioritized DDQN (*)	140%	128%
Dueling DDQN (*)	151%	117%
A3C (*)	-	116%
Noisy DQN	118%	102%
Distributional DQN	185%	125%
Rainbow	**231%**	**153%**

学习速率.

与原始 DQN 设置相同，我们在单个 GPU 上运行每个智能体。达到与 DQN 最终性能相当所需的 7M 帧对应的时间不到 10 小时。完整运行 200M 帧大约需要 10 天，并且在所有讨论的变体之间差异小于 20%。文献中包含了许多替代的训练设置，它们通过利用并行性来提高以实际运行时间衡量的性能，例如 Nair et al. (2015), Salimans et al. (2017), and Mnih et al. (2016).。在如此不同的硬件/计算资源之间进行性能比较并非易事，因此我们仅专注于算法层面的变体，从而实现了可比性。虽然我们认为可扩展性和并行化的问题同样重要且互为补充，但我们将其留待未来工作中研究。

消融实验.

由于 Rainbow 将多种不同的思想整合到一个智能体中，我们进行了额外的实验，以理解各个组成部分在这一特定组合中的贡献。

首先，我们执行了消融研究。在每个消融实验中，我们从完整的 Rainbow 组合中移除一个组件，并在所有 Atari 游戏上训练所得的智能体。图 .9 比较了 Rainbow 与六个消融变体的中位数归一化得分。图 .8 展示了这些消融在不同的人类归一化性能阈值下的更详细表现，而图 .10 则显示了每个消融在所有游戏中的平均增益或损失。

在 Rainbow 的组成部分中，优先经验回放和多步学习是两个最关键的组件，因为移除任一组件都会导致中位性能显著下降。不出所料，去除这两个组件都会削弱早期性能。更令人意外的是，去除多步学习同样会降低最终性能。进一步观察具体游戏（见图 .10），我们发现这两个组件几乎在所有游戏中都带来了一致的改进（在 57 个游戏中有 53 个 Rainbow 的表现优于对应的消融变体）。

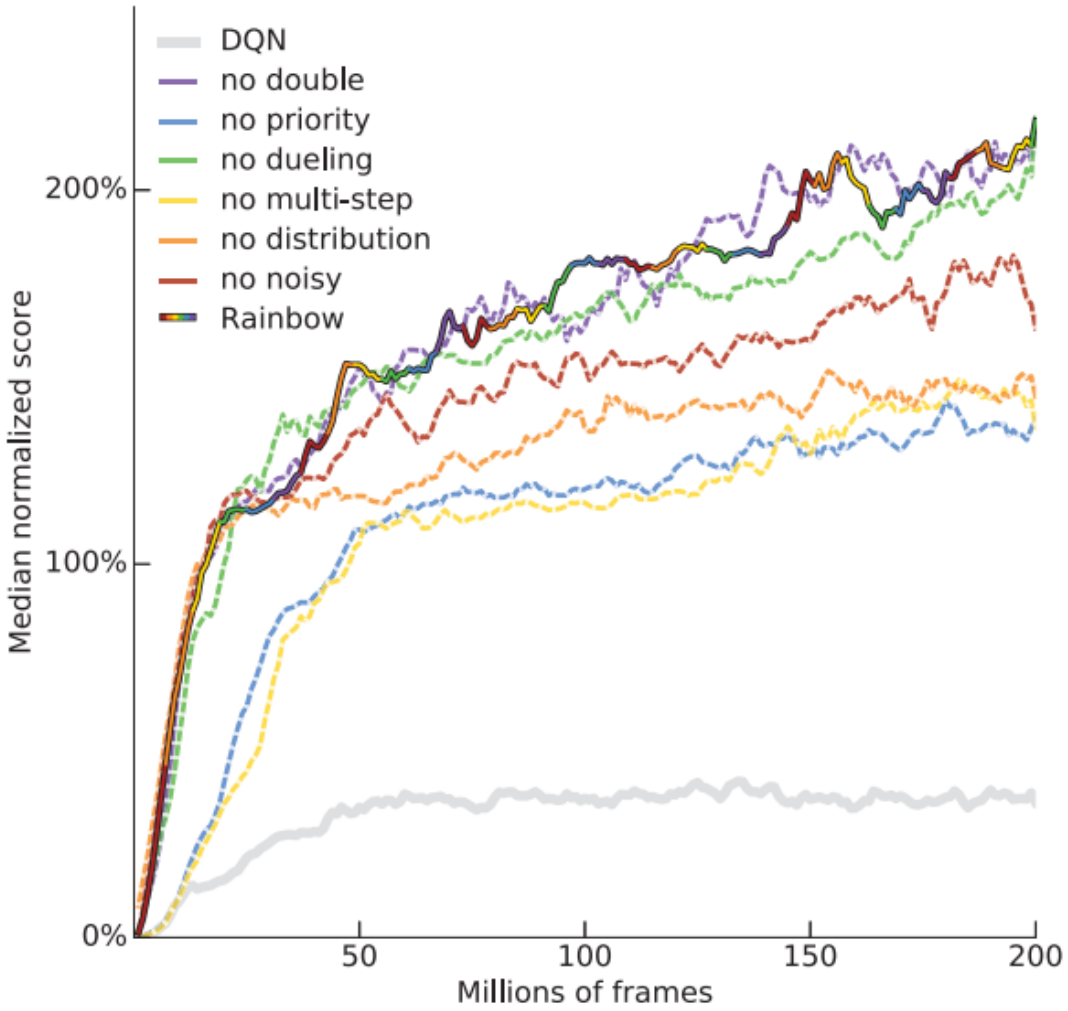


图 0.35　本图展示了 57 款雅达利游戏在时间维度上的中位数人类归一化性能表现。我们将集成智能体（彩虹色）与 DQN 算法（灰色）进行对比，并以虚线形式呈现六种消融实验结果。所有曲线均采用 10 点移动平均法进行平滑处理。

分布的 Q-learning 的重要性紧随其后。值得注意的是，在早期训练中几乎没有差异（如图 .9 所示），在前 4000 万帧内，去除分布的消融变体表现与完整智能体相当。然而，在没有分布的情况下，智能体的性能随后开始落后。当我们将结果按照人类水平进行分层比较时（图 .8），可以看到分布消融主要在接近或超过人类水平的游戏中落后。

从中位性能来看，包含 Noisy Nets 的智能体表现更佳；而当移除 Noisy Nets，并将探索交由传统的 ϵ-greedy 机制时，总体性能更差（见图 .9 中的红线）。虽然去除 Noisy Nets 在多个游戏中导致了显著的性能下降，但在少数游戏中也带来了轻微提升（见图 .10）。

总体而言，从完整 Rainbow 中移除 Dueling 网络并未带来显著差异。然而，中位数得分掩盖了不同游戏间的差异（见图 .10）。图 .8 显示，Dueling 在部分超过人

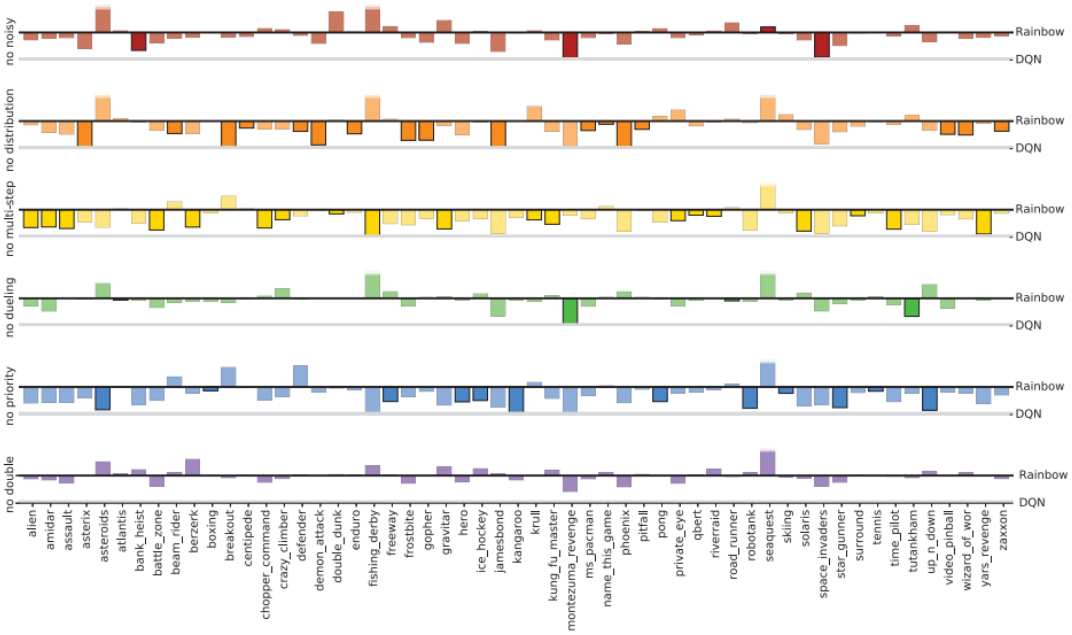


图 0.36　消融剂性能下降分析：该指标以曲线下面积为基准，通过与 Rainbow 和 DQN 的标准化对比呈现。在 Venture 和 Bowling 两款游戏中，DQN 表现优于 Rainbow 的数据被排除在外。每局游戏中导致性能降幅最大的消融剂均被高亮标注。优先级设置与多步骤辅助功能在各局游戏中基本保持一致，而其他组件的影响则因局而异。

类水平的游戏（性能 >200%）中可能带来了一定提升，而在部分低于人类水平的游戏（性能 >20%）中则可能造成性能下降。

类似地，在 Double Q-learning 的情形下，观察到的中位性能差异（图 .9）有限，其作用可能因游戏而异（见图 .10）。为了进一步探究 Double Q-learning 的作用，我们比较了已训练智能体的预测值与基于裁剪奖励计算的实际折扣回报。比较 Rainbow 与移除 Double Q-learning 的智能体时，我们发现实际回报常常超过 10，从而超出了分布的支持范围（$[-10, 10]$）。这导致了回报被低估，而非传统的过高估计。我们推测，将取值裁剪到有限范围内抵消了 Q-learning 的高估偏差。然而，需要注意的是，如果分布的支持范围被扩展，Double Q-learning 的重要性可能会增加。

6 论述

我们已经证明，多个对 DQN 的改进方法可以成功地整合到一个学习算法中，并实现了当前最优的性能。此外，我们还展示了，在该集成算法中，除一个组件外，其余所有组件都提供了明确的性能提升。仍然有许多算法组件尚未纳入，而它们是未来在集成智能体实验中的有希望的候选对象。下面我们讨论其中的一些方向。

我们主要聚焦于 Q-learning 家族的基于价值的方法，但类似的思想也可能对基于策略的强化学习算法有益，例如 TRPO (Schulman et al. 2015) 或者 actor-critic

方法 (Mnih et al. 2016; O’Donoghue et al. 2016)。已有许多算法利用数据序列来提升学习效率。例如，Optimality tightening (He et al. 2016) 使用多步回报来构造额外的不等式约束，而不仅仅是替换 Q-learning 中的 1 步目标；资格迹 (Sutton 1988) 则允许对 n 步回报进行软组合。然而，这类序列方法每次更新所需的计算量通常多于 Rainbow 使用的多步目标。此外，优先经验回放与序列数据的结合仍然是一个未解决的问题。

Episodic control (Blundell et al. 2016) 同样关注数据效率，并且在某些领域显示出很高的效果。它通过使用情节记忆作为一种互补的学习系统来改进早期学习，从而能够立即重现成功的动作序列。

除了 Noisy Nets，还有许多探索方法被提出：例如自举 DQN (Osband et al. 2016)、内在动机 (Stadie, Levine, and Abbeel 2015) 以及基于计数的探索 (Bellemare et al. 2016)。将这些方法与 Rainbow 相结合，是一个有前景的研究方向。

本文聚焦于核心学习更新，而未探索替代的计算架构。例如，A3C (Mnih et al. 2016)、Gorila (Nair et al. 2015) 或者进化策略 (Salimans et al. 2017) 这类方法通过在环境的并行副本上进行异步学习，可以在实际时间上加速学习，尽管这会以数据效率为代价。

分层强化学习（HRL）也已在多个复杂 Atari 游戏中成功应用。例如，h-DQN (Kulkarni et al. 2016a) 和 Feudal Networks (Vezhnevets et al. 2017) 都是其中的代表性工作。

状态表示也可以通过辅助任务得到改进，例如像素或特征控制 (Jaderberg et al. 2016)、监督预测 (Dosovitskiy and Koltun 2016) 或者 successor features (Kulkarni et al. 2016b)。

为了公平地将 Rainbow 与基线进行比较，我们遵循了常见的修正方法，包括帧堆叠、奖励裁剪以及固定的动作重复。这些方法可能被更为合理的技术所替代。例如，循环网络 (Hausknecht and Stone 2015) 可以学习时间上的状态表示，从而取代帧堆叠；Pop-Art (van Hasselt et al. 2016) 可以直接从原始奖励中学习；而精细化的动作重复 (Sharma, Lakshminarayanan, and Ravindran 2017) 则能够学习动作重复的次数。总体而言，我们认为将真实的游戏环境直接暴露给智能体，是未来研究的一个很有前景的方向。

参考文献

[1] Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res. (JAIR)*, 47:253–279.

[2] Bellemare, M. G.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *NIPS*.

[3] Bellemare, M. G.; Dabney, W.; and Munos, R. (2017). A distributional perspective on reinforcement learning. In *ICML*.

[4] Blundell, C.; Uria, B.; Pritzel, A.; Li, Y.; Ruderman, A.; Leibo, J. Z.; Rae, J.; Wierstra, D.; and Hassabis, D. (2016). Model-Free Episodic Control. *ArXiv e-prints*.

[5] Dosovitskiy, A., and Koltun, V. (2016). Learning to act by predicting the future. *CoRR*, abs/1611.01779.

[6] Fortunato, M.; Azar, M. G.; Piot, B.; Menick, J.; Osband, I.; Graves, A.; Mnih, V.; Munos, R.; Hassabis, D.; Pietquin, O.; Blundell, C.; and Legg, S. (2017). Noisy networks for exploration. *CoRR*, abs/1706.10295.

[7] Hausknecht, M., and Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. *arXiv preprint*, arXiv:1507.06527.

[8] He, F. S.; Liu, Y.; Schwing, A. G.; and Peng, J. (2016). Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *CoRR*, abs/1611.01606.

[9] Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, abs/1611.05397.

[10] Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. In *Proc. of ICLR*.

[11] Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. B. (2016a). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR*, abs/1604.06057.

[12] Kulkarni, T. D.; Saeedi, A.; Gautam, S.; and Gershman, S. J. (2016b). Deep successor reinforcement learning. *arXiv preprint*, arXiv:1606.02396.

[13] Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321.

[14] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

[15] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[16] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *Proc. of ICML*.

[17] Nair, A.; Srinivasan, P.; Blackwell, S.; Alcicek, C.; Fearon, R.; De Maria, A.; Panneershelvam, V.; Suleyman, M.; Beattie, C.; Petersen, S.; Legg, S.; Mnih, V.; Kavukcuoglu, K.; and Silver, D. (2015). Massively parallel methods for deep reinforcement learning. *arXiv preprint*, arXiv:1507.04296.

[18] O’Donoghue, B.; Munos, R.; Kavukcuoglu, K.; and Mnih, V. (2016). Pgq: Combining policy gradient and q-learning. *CoRR*, abs/1611.01626.

[19] Osband, I.; Blundell, C.; Pritzel, A.; and Roy, B. V. (2016). Deep exploration via bootstrapped DQN. In *NIPS*.

[20] Salimans, T.; Ho, J.; Chen, X.; and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *CoRR*, abs/1703.03864.

[21] Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. (2015). Prioritized experience replay. In *Proc. of ICLR*.

[22] Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.; and Abbeel, P. (2015). Trust region policy optimization. In *Proc. of ICML*, 1889–1897.

[23] Sharma, S.; Lakshminarayanan, A. S.; and Ravindran, B. (2017). Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint*, arXiv:1702.06054.

[24] Stadie, B. C.; Levine, S.; and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *CoRR*, abs/1507.00814.

[25] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

[26] Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA.

[27] Tieleman, T., and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[28] van Hasselt, H.; Guez, A.; Hessel, M.; Mnih, V.; and Silver, D. (2016). Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, 4287–4295.

[29] van Hasselt, H.; Guez, A.; and Silver, D. (2016). Deep reinforcement learning with double Q-learning. In *Proc. of AAAI*, 2094–2100.

[30] van Hasselt, H. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems*, 2613–2621.

[31] Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. *CoRR*, abs/1703.01161.

[32] Wang, Z.; Schaul, T.; Hessel, M.; van Hasselt, H.; Lanctot, M.; and de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proc. of ICML*, 1995–2003.

异步深度强化学习方法[1,2]

摘要

我们提出了一种概念简单且轻量级的深度强化学习框架，该框架利用异步梯度下降来优化深度神经网络控制器。我们给出了四种标准强化学习算法的异步变体，并表明并行的 actor learner 在训练过程中起到了稳定作用，使所有四种方法都能成功地训练神经网络控制器。其中表现最佳的算法是异步优势 actor critic，它在 Atari 游戏领域中超越了当时的最新水平，并且仅在一台多核 CPU 上训练了一半时间，无需使用 GPU。此外，我们还展示了异步优势 actor critic 在一系列连续运动控制任务以及一项仅通过视觉输入在随机三维迷宫中导航的新任务上同样取得了成功。

1 引言

深度神经网络能够提供丰富的表示，从而使强化学习（RL）算法能够有效地执行。然而，先前人们认为将简单的在线强化学习算法与深度神经网络结合在本质上是难以稳定的。为此，研究者们提出了多种解决方案以稳定算法[1,1,3,5,7]。这些方法共享一个核心思想：在线强化学习智能体所遇到的数据序列是非平稳的，且在线更新之间存在强相关性。通过将智能体的数据存储在经验回放（experience replay）记忆中，可以将数据进行批处理[1,5]或从不同时间步中随机采样[1,3,7]。通过对记忆进行聚合，可以减少非平稳性并降低更新之间的相关性，但与此同时，这些方法也被限制在离轨（off-policy）强化学习中。

基于经验回放的深度强化学习算法在诸如 Atari 2600 等挑战性领域中取得了前所未有的成功。然而，经验回放也存在若干缺点：它在每次真实交互中消耗更多的内存与计算资源；并且它要求使用离轨学习算法，这些算法能够从由旧策略生成的数据中进行更新。

在本文中，我们提出了一种截然不同的深度强化学习范式。我们不再使用经验回放，而是异步地在多个环境中并行执行多个智能体。这种并行化同样能够将智能体的数据去相关化为一个更平稳的过程，因为在任意给定时间步，并行智能体将经历各种不同的状态。这个简单的想法使得更大范围的基础在轨（on-policy）强

[1]原文：Asynchronous Methods for Deep Reinforcement Learning, Mnih et al, 2016. Algorithm: A3C.

[2]译者：陈子涵。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

化学习算法（如 Sarsa、n 步方法和 actor-critic 方法）以及离轨算法（如 Q-learning）能够稳健且有效地应用于深度神经网络。

我们提出的并行强化学习范式还具有若干实际优势。尽管先前的深度强化学习方法严重依赖于专用硬件（如 GPU）[3,5,7] 或大规模分布式架构[3]，我们的实验仅在一台配备标准多核 CPU 的单机上运行。在应用于多种 Atari 2600 游戏时，异步强化学习方法在许多游戏中取得了更好的结果，训练时间远少于先前的 GPU 算法，且资源消耗远低于大规模分布式方法。我们提出的最佳方法：异步优势 actor-critic（A3C）还在多种连续运动控制任务中表现出色，并学会了仅通过视觉输入在 3D 迷宫中进行探索的通用策略。我们相信 A3C 在 2D 与 3D 游戏、离散与连续动作空间中的成功，以及其训练前馈与循环智能体的能力，使其成为迄今为止最通用且最成功的强化学习智能体。

2 相关工作

分布式强化学习架构：在[3] 提出的 Gorila（General Reinforcement Learning Architecture）框架中，首次在分布式环境中实现了强化学习智能体的异步训练。在 Gorila 中，每个进程包含一个 actor，它在自身的环境副本中执行动作，拥有一个独立的经验回放记忆，以及一个 learner，从回放记忆中采样数据并计算 DQN 损失函数关于策略参数的梯度。这些梯度被异步地发送到一个中央参数服务器，该服务器更新一个全局模型副本。更新后的策略参数会以固定间隔发送回各个 actor-learner。通过使用 100 个 actor-learner 进程和 30 个参数服务器实例（共 130 台机器），Gorila 在 49 款 Atari 游戏上显著超越了 DQN 的表现，在许多游戏中，其达到 DQN 水平的速度快了 20 倍以上。我们也注意到，[24] 也提出了一种类似的并行化 DQN 的方法。

早期并行强化学习研究：在更早的研究中，[8] 将 MapReduce 框架应用于批量强化学习方法的并行化，采用的是线性函数逼近。其并行化主要用于加速大型矩阵运算，而非用于并行经验收集或稳定学习过程。[9] 提出了一种并行化的 Sarsa 算法，使用多个独立的 actor-learner 来加速训练。每个 actor-learner 独立学习，并定期将权重更新发送给其他智能体，采用点对点通信方式。

异步优化理论：[10] 研究了异步优化设置下 Q-learning 的收敛性质。结果表明，只要过时的信息最终被丢弃，并满足其他一些技术性假设，即使部分信息是过时的，Q-learning 仍然保证收敛。更早地，[11] 也研究了分布式动态规划的相关问题。

进化算法与并行化：另一个相关领域是进化算法，这类方法通常易于通过将适应度评估分布到多个机器或线程上进行并行化[12]。最近，这类并行进化方法已被应用于一些视觉强化学习任务。例如，[13] 通过在 8 个 CPU 核心上并行执行适应度评

估，演化出用于 TORCS 驾驶模拟器的卷积神经网络控制器。

3 强化学习背景

我们考虑标准的强化学习设定：智能体在多个离散时间步中与一个环境 E 进行交互。在每个时间步 t，智能体接收到一个状态 s_t，并根据其策略 π 从动作集合 A 中选择一个动作 a_t，其中 π 是从状态到动作的映射。随后，智能体接收到下一个状态 s_{t+1} 和一个标量奖励 r_t。该过程持续进行，直到智能体到达终止状态，然后过程重新开始。

回报（return）定义为从时间步 t 开始的折扣累积奖励：

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

其中 $\gamma \in (0, 1]$ 是折扣因子。智能体的目标是从每个状态 s_t 最大化期望回报。

动作价值函数 $Q^\pi(s, a)$ 表示在状态 s 下选择动作 a 并遵循策略 π 的期望回报：

$$Q^\pi(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a].$$

最优价值函数 $Q^*(s, a)$ 是所有策略中最大的动作价值：

$$Q^*(s, a) = \max_\pi Q^\pi(s, a).$$

状态价值函数 $V^\pi(s)$ 表示从状态 s 出发遵循策略 π 的期望回报：

$$V^\pi(s) = \mathbb{E}[R_t \mid s_t = s].$$

在无模型的价值函数方法中，动作价值函数通常使用函数逼近器（如神经网络）进行表示。设 $Q(s, a; \theta)$ 为参数为 θ 的近似动作价值函数。参数更新可来源于多种强化学习算法。例如，Q-learning 旨在直接逼近最优动作价值函数：

$$Q^*(s, a) \approx Q(s, a; \theta).$$

在一步 Q-learning 中，参数 θ 通过最小化以下损失函数序列进行学习：

$$L_i(\theta_i) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)^2\right],$$

其中 s' 是在状态 s 执行动作 a 后到达的下一个状态。一步方法的缺点是，获得一个奖励 r 仅直接影响导致该奖励的状态-动作对的价值，其他状态-动作对的价值仅通过间接方式更新。这可能使得学习过程变慢，因为需要大量更新才能将奖励传播到相关的前序状态和动作。

一种加速奖励传播的方法是使用 n 步回报。在 n 步 Q-learning 中，$Q(s,a)$ 向 n 步回报更新：

$$R_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_a Q(s_{t+n}, a).$$

在 n 步 Q-learning 中，参数 θ 通过最小化以下损失函数序列进行学习：

$$L_i(\theta_i) = \mathbb{E}\left[\left(R_t^{(n)} - Q(s_t, a_t; \theta_i)\right)^2\right],$$

这样，单个奖励 r 将直接影响前 n 个状态-动作对的价值，使得奖励传播更加高效。

与基于价值的方法不同，基于策略的无模型方法直接对策略 $\pi(a|s;\theta)$ 进行参数化，并通过（通常是近似的）梯度上升来最大化期望回报 $\mathbb{E}[R_t]$。REINFORCE 算法族[25] 是这类方法的一个例子。标准 REINFORCE 算法通过以下梯度估计更新策略参数：

$$\nabla_\theta \log \pi(a_t|s_t;\theta) R_t,$$

为了减小方差，通常会引入一个基线函数 $b(s_t)$，使得梯度估计变为：

$$\nabla_\theta \log \pi(a_t|s_t;\theta)(R_t - b(s_t)).$$

通常使用学习到的价值函数估计作为基线 $b_t(s_t) \approx V^\pi(s_t)$，从而显著降低策略梯度估计的方差。当使用近似价值函数作为基线时，用于缩放策略梯度的量 $R_t - b_t$ 可以看作是在状态 s_t 下动作 a_t 的优势估计，即 $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$，因为 R_t 是 $Q^\pi(a_t, s_t)$ 的估计，而 b_t 是 $V^\pi(s_t)$ 的估计。这种方法可以看作是一种 actor-critic 架构，其中策略 π 是 actor，基线 b_t 是 critic[15,26]。

4 异步强化学习框架

我们提出了多种异步的强化学习算法变体，包括：一步 Sarsa、一步 Q-learning、n 步 Q-learning 和优势 actor-critic。设计这些方法的目标是找到能够在不依赖大量资源的情况下可靠地训练深度神经网络策略的强化学习算法。尽管这些底层强化学习方法存在差异（例如，actor-critic 是一种在轨搜索方法，而 Q-learning 是离轨

的价值方法），我们采用了两个主要思想，使得所有四种算法都能在我们的设计目标下实际可用。

首先，我们使用了异步的 actor-learner，类似于 Gorila 框架[3]，但我们不使用独立的机器和参数服务器，而是使用单台机器上的多个 CPU 线程。将 learner 限制在单台机器上消除了发送梯度和参数的通信开销，并使我们能够使用 Hogwild! 风格的更新方式进行训练[4]。（Hogwild! 算法是一种用于深度学习的无锁并行随机梯度下降（SGD）算法，它允许多个计算节点异步地更新共享模型参数，而不需要使用锁机制来同步这些更新。这种方法可以减少计算节点之间的等待时间，提高并行计算的效率。Hogwild! 算法特别适用于大规模稀疏数据集，因为它通过引入随机性来加速大规模并行机器学习任务。）

其次，我们观察到，多个并行运行的 actor-learner 很可能在探索环境的不同部分。此外，还可以在每个 actor-learner 中显式地使用不同的探索策略以最大化这种多样性。通过在不同线程中运行不同的探索策略，多个 actor-learner 并行地应用在线更新时，对参数的整体变化在时间上可能比一个智能体单独应用更新时更不相关。因此，我们不使用经验回放，而是依赖于并行的 actor 使用不同的探索策略来起到稳定训练的作用，这在 DQN 训练算法中是由经验回放完成的。

除了稳定学习之外，使用多个并行的 actor-learner 还具有多个实际好处。首先，我们获得了与并行 actor-learner 数量大致成线性关系的训练时间减少。其次，由于我们不再依赖经验回放来稳定学习，因此可以使用在轨强化学习方法（如 Sarsa 和 actor-critic）来稳定地训练神经网络。

异步一步 Q 学习

异步一步 Q 学习的伪代码如算法 1 所示。每个线程与自己的环境副本交互，并在每一步计算 Q 学习损失的梯度。我们使用一个共享且缓慢变化的目标网络来计算 Q 学习损失，这在 DQN 训练方法中被提出。我们还积累多个时间步的梯度，然后才应用它们，这类似于使用小批量。这减少了多个 actor learner 相互覆盖更新的机会。积累多个步骤的更新还提供了在计算效率和数据效率之间进行权衡的一些能力。

算法 4 异步一步 Q 学习 - 每个 actor-learner 线程的伪代码

假设全局共享 θ, θ^-, 和计数器 $T = 0$.
初始化线程步计数器 $t \leftarrow 0$
初始化目标网络权重 $\theta^- \leftarrow \theta$
初始化网络梯度 $d\theta \leftarrow 0$
获取初始状态 s
repeat
    基于 $Q(s, a; \theta)$ 使用 ϵ-贪婪策略采取动作 a
    接收新状态 s' 和奖励 r

$$y = \begin{cases} r & \text{若 } s' \text{ 为终止状态} \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{若 } s' \text{ 为非终止状态} \end{cases}$$

    累积关于 θ 的梯度：$d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
    $s \leftarrow s'$
    $T \leftarrow T + 1$ 和 $t \leftarrow t + 1$
    if $T \mod I_{\text{target}} == 0$ **then**
        更新目标网络 $\theta^- \leftarrow \theta$
    end if
    if $t \mod I_{\text{asyncUpdate}} == 0$ 或 s 是终止状态 **then**
        使用 $d\theta$ 异步更新 θ
        清除梯度 $d\theta \leftarrow 0$
    end if
until $T > T_{\text{max}}$

最后，我们发现给每个线程一个不同的探索策略有助于提高鲁棒性。以这种方式增加探索的多样性通常也能通过更好的探索来提高性能。虽然有很多方法可以使探索策略不同，但我们实验了使用 ϵ-贪婪探索，其中 ϵ 由每个线程定期从某个分布中采样。

异步一步 Sarsa

异步一步 Sarsa 算法与异步一步 Q 学习相同，如算法 1 所示，只是它对 $Q(s, a)$ 使用不同的目标值。一步 Sarsa 使用的目标值是 $r + \gamma Q(s', a'; \theta^-)$，其中 a' 是在状态 s' 采取的动作（Rummery & Niranjan, 1994; Sutton & Barto, 1998）。我们再次使用目标网络，并在多个时间步上累积更新来稳定学习。

异步 n 步 Q 学习

我们变体的多步 Q 学习的伪代码如补充算法 S1 所示。该算法有些不寻常，因为它通过显式计算 n 步回报来操作前向视图，而不是使用更常见的后向视图，后者通过资格迹（eligibility traces）（Sutton & Barto, 1998）隐式地组合不同的回报。

我们发现，使用前向视图在训练神经网络时更容易，特别是当使用基于动量的梯度下降方法和时间反向传播时。为了计算单个更新，算法首先使用其探索策略选择动作，直到达到 $t_{\max}$ 步或达到终止状态。这个过程导致智能体从环境中接收多达 $t_{\max}$ 个奖励，因为它自上次更新以来。然后算法计算自上次更新以来遇到的每个状态-动作对的 n 步 Q 学习更新的梯度。每个 n 步更新使用最长可能的 n 步回报，导致最后一个状态进行一步更新，倒数第二个状态进行两步更新，依此类推，总共多达 $t_{\max}$ 个更新。累积的更新在单个梯度步骤中应用。

异步优势 Actor-Critic

我们称之为异步优势 Actor-Critic (A3C) 的算法维护策略 $\pi(a_t|s_t;\theta)$ 和价值函数 $V(s_t;\theta_v)$ 的估计。与我们的 n 步 Q 学习变体类似，我们的 Actor-Critic 变体也操作前向视图，并使用相同的 n 步回报混合来更新策略和价值函数。策略和价值函数在每 $t_{\max}$ 个动作或达到终止状态后更新一次。算法执行的更新可以看作是

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta')A(s_t,a_t;\theta,\theta_v),$$

其中 $A(s_t,a_t;\theta,\theta_v)$ 是优势函数的估计，由下式给出

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v),$$

其中 k 可以因状态而异，上限由 $t_{\max}$ 决定。算法的伪代码在补充算法 S2 中给出。

与基于价值的方法一样，我们依赖于并行 actor-learner 和累积更新来提高训练稳定性。注意，尽管策略参数 θ 和价值函数参数 θ_v 在一般情况下是分开的，我们实际上总是共享一些参数。我们通常使用一个卷积神经网络，该网络有一个 softmax 输出用于策略 $\pi(a_t|s_t;\theta)$ 和一个线性输出用于价值函数 $V(s_t;\theta_v)$，所有非输出层都是共享的。

我们还发现，将策略 π 的熵添加到目标函数中通过阻止过早收敛到次优确定性策略来改善探索。这种技术最初由 Williams & Peng (1991) 提出，他们发现它在需要层次化行为的任务上特别有帮助。包括熵正则项的完整目标函数相对于策略参数的梯度形式为

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta')(R_t - V(s_t;\theta_v)) + \beta\nabla_{\theta'} H(\pi(s_t;\theta')),$$

其中 H 是熵。超参数 β 控制熵正则项的强度。

优化

我们在我们的异步框架中研究了三种不同的优化算法——带动量的 SGD、无共享统计量的 RMSProp 和共享统计量的 RMSProp。我们使用标准的非中心化 RMSProp 更新，由下式给出

$$g = \alpha g + (1-\alpha)\Delta\theta^2 \quad \text{和} \quad \theta \leftarrow \theta - \eta\frac{\Delta\theta}{\sqrt{g+\epsilon}},$$

其中所有操作都是逐元素执行的。在 Atari 2600 游戏的一个子集上的比较表明，共享统计量 g 的 RMSProp 比其他两种方法更稳健。方法和比较的完整细节包含在补充材料第 1 节中。

5 实验

我们使用四个不同的平台来评估所提出框架的性能。我们主要在 Arcade Learning Environment（ALE）[21] 上进行实验，该环境提供了 Atari 2600 游戏的模拟器，是强化学习算法最常用的基准测试环境之一。我们使用 Atari 领域与当前最先进的方法进行比较[3,3,5,7,8]，并对所提出方法的稳定性和可扩展性进行详细分析。我们还使用 TORCS 3D 赛车模拟器[22] 进行进一步比较。此外，我们还在两个额外领域上评估了 A3C 算法：MuJoCo 物理引擎和 Labyrinth 3D 迷宫环境。实验设置的详细信息请参见补充材料。

Atari 2600 游戏

我们首先在一组 Atari 2600 游戏上展示新方法的训练速度。图 1 比较了在 Nvidia K40 GPU 上训练的 DQN 算法与使用 16 个 CPU 核心训练的异步方法在五款 Atari 2600 游戏上的学习速度。结果表明，我们提出的四种异步方法都能够成功地在 Atari 领域训练神经网络控制器。异步方法的学习速度通常快于 DQN，在某些游戏上学习速度显著更快，而仅使用了 16 个 CPU 核心。此外，结果还表明，在某些游戏中，n 步方法比一步方法学习得更快。总体而言，基于策略的优势 actor-critic 方法（A3C）显著优于所有三种基于价值的方法。

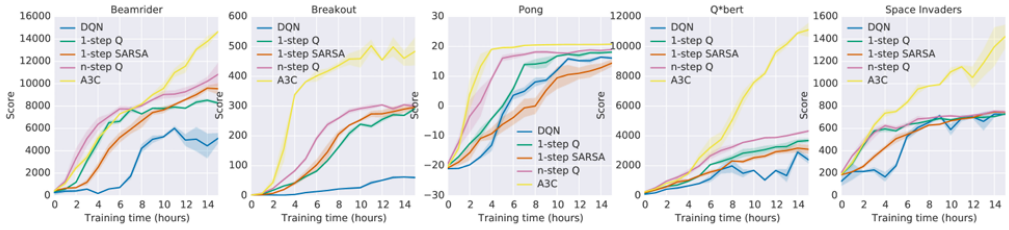


图 0.37　在五个 Atari 2600 游戏上，DQN 与异步方法的学习速度对比。DQN 在单个 Nvidia K40 GPU 上训练，而异步方法使用 16 个 CPU 核心训练。曲线为 5 次运行的平均值。对于 DQN，使用固定超参数和不同随机种子；对于异步方法，从 50 组实验中选出表现最好的 5 组模型，学习率从 LogUniform($10^{-4}, 10^{-2}$) 中采样，其余超参数固定。

随后，我们在 57 款 Atari 游戏上评估了异步优势 actor-critic（A3C）算法。为了与当前最先进的方法进行比较，我们基本遵循了[7] 的训练与评估协议。具体而言，我们在 6 款 Atari 游戏（Beamrider、Breakout、Pong、Q*bert、Seaquest 和 Space Invaders）上进行超参数搜索（学习率和梯度裁剪范围），然后将所有超参数固定，应用于全部 57 款游戏。我们训练了两种智能体：一种是前馈网络（与[3,3,7] 相同的架构），另一种是循环网络（在最后一个隐藏层后增加 256 个 LSTM 单元）。我们使用最终网络权重进行评估，以使结果与原始结果更具可比性。

方法	训练时间	平均得分	中位得分
DQN	8 天 GPU	121.9%	47.5%
Gorila	4 天, 100 机器	215.2%	71.3%
D-DQN	8 天 GPU	332.9%	110.9%
Dueling D-DQN	8 天 GPU	343.8%	117.1%
Prioritized DQN	8 天 GPU	463.6%	127.6%
A3C, FF	1 天 CPU	344.1%	68.2%
A3C, FF	4 天 CPU	496.8%	116.6%
A3C, LSTM	4 天 CPU	623.0%	112.6%

表 19　在 57 款 Atari 游戏上使用人类起始评估指标的平均和中位人类归一化得分。补充表格 SS1 展示了所有游戏的原始分数

从表 1 可以看出，A3C 在仅使用 16 个 CPU 核心、无 GPU 的情况下，在 4 天内达到了优于其他方法的平均人类归一化得分，而其它方法通常需要在 GPU 上训练 8 至 10 天。值得注意的是，仅训练 1 天的 A3C 前馈模型就已达到了与 Dueling Double DQN 相当的平均得分，并接近 Gorila 的中位得分。我们还指出，Double DQN 和 Dueling Double DQN 中的许多改进也可以应用于本文提出的一步 Q 和 n 步 Q 方法中，可能带来类似的性能提升。

TORCS 赛车模拟器

我们还在 TORCS 3D 赛车游戏上比较了四种异步方法的性能[22]。与 Atari 2600 游戏相比，TORCS 不仅具有更真实的图形，还要求智能体学习所控制赛车的动力学特性。在每一步，智能体仅接收当前帧的 RGB 图像作为输入，以及一个与其沿赛道中心线速度成正比的奖励信号。

我们使用了与 Atari 实验相同的神经网络架构（详见补充材料第 2 节）。我们在四种不同的设置下进行实验：智能体控制慢车（有/无对手）与快车（有/无对手）。完整结果见补充图 S2。A3C 是表现最佳的方法，在所有四种配置下，经过约 12 小时训练后，其得分达到了人类测试者得分的约 75% 至 90%。展示 A3C 学习到的驾驶行为的视频可在以下链接观看：`https://youtu.be/0xo1Ldx3L5Q`。

连续动作控制：MuJoCo 物理模拟器

我们还在一组连续动作任务中评估了所提出方法。具体而言，我们考察了一组具有接触动力学的刚体物理领域，任务包括多种操作与运动控制问题。这些任务使用 MuJoCo 物理引擎进行模拟[23]。

由于基于价值的方法（如 Q-learning）不易扩展到连续动作空间，我们仅评估了异步优势 actor-critic（A3C）算法。在所有任务中，无论是以物理状态还是像素作为输入，A3C 都能在不到 24 小时的时间内找到良好的策略，通常在几小时内就能收敛。部分成功策略的可视化视频可在以下链接观看：`https://youtu.be/Ajjc08-iPx8`。有关实验的更多细节请参见补充材料第 3 节。

Labyrinth 3D 迷宫环境

我们还在一个新的 3D 环境——Labyrinth 中进行了 A3C 的实验。具体任务中，智能体需要学习在随机生成的迷宫中寻找奖励。在每个 episode 开始时，智能体被放置在一个新的随机生成的迷宫中，迷宫由房间和走廊组成。迷宫中包含两种类型的对象：苹果和传送门。拾取一个苹果可获得奖励 1，进入传送门可获得奖励 10，之后智能体会在迷宫中的新随机位置重生，所有苹果也会重新生成。每个 episode 持续 60 秒，结束后开始新的 episode。

智能体的目标是尽可能多地收集分数，而最优策略是首先找到传送门，然后在每次重生后反复前往该传送门。该任务比 TORCS 驾驶任务更具挑战性，因为智能体在每个 episode 中都面临全新的迷宫，必须学习一种通用的探索策略。

我们仅使用 84×84 的 RGB 图像作为输入，训练了一个 A3C LSTM 智能体。最终平均得分约为 50，表明该智能体仅通过视觉输入就学会了在随机 3D 迷

宫中进行有效探索的策略。展示智能体探索未见迷宫的视频可在以下链接观看：https://youtu.be/nMR5mjCFZCw。

可扩展性与数据效率

我们通过分析训练时间和数据效率随并行 actor-learner 数量的变化来评估所提出框架的有效性。在使用多个 worker 并行更新共享模型时，理想情况下，对于给定任务和算法，达到某一性能所需的训练步数应不随 worker 数量变化。因此，优势主要来自于在相同墙钟时间内处理更多数据，以及可能的探索改善。

表 2 显示了在 7 款 Atari 游戏上，使用不同数量并行 actor-learner 所获得的训练加速比。我们测量了每种方法和线程数达到固定参考得分所需的时间，并计算加速比。结果表明，所有四种方法都通过使用多个 worker 实现了显著的加速，16 个线程带来了至少一个数量级的加速。这验证了我们提出的框架在并行 worker 数量增加时具有良好的可扩展性，能够高效利用计算资源。

方法	1 线程	2 线程	4 线程	8 线程	16 线程
1-step Q	1.0	3.0	6.3	13.3	24.1
1-step SARSA	1.0	2.8	5.9	13.1	22.1
n-step Q	1.0	2.7	5.9	10.7	17.2
A3C	1.0	2.1	3.7	6.9	12.5

表 20　该表展示了每种方法和线程数量在七款 Atari 游戏上的平均训练加速比。为了计算单个游戏的训练加速，我们测量了使用每种方法和线程数量达到固定参考得分所需的时间。从使用 n 个线程的游戏上获得的加速比定义为使用一个线程达到固定参考得分所需的时间除以使用 n 个线程达到参考得分所需的时间。该表显示了在七款 Atari 游戏（Beamrider, Breakout, Enduro, Pong, Q*bert, Seaquest, 和 Space Invaders）上的平均加速比。

令人惊讶的是，异步一步 Q-learning 和 Sarsa 算法表现出超线性加速，这无法用单纯的计算增益解释。我们观察到，一步方法（1 步 Q 和 1 步 Sarsa）在使用更多并行 actor-learner 时，通常需要更少的数据就能达到特定得分。我们认为，这是由于多线程在减少一步方法偏差方面的积极作用。图 3 和图 4 更清晰地展示了这些效应，分别显示了不同线程数下平均得分随训练帧数和墙钟时间的变化。

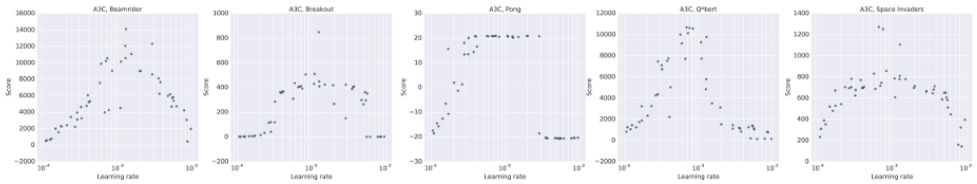


图 0.38　A3C 在 5 款游戏（Beamrider、Breakout、Pong、Q*bert、Space Invaders）上使用 50 组不同学习率与随机初始化的得分分布散点图。每款游戏均存在较宽的学习率区间，使得所有初始化都能获得良好得分，表明 A3C 对学习率与随机初始化具有良好的鲁棒性。

稳定性与鲁棒性

最后，我们分析了四种异步算法的稳定性和鲁棒性。对于每种算法，我们在 5 款游戏（Breakout、Beamrider、Pong、Q*bert、Space Invaders）上使用 50 组不同的学习率和随机初始化进行训练。图 2 显示了 A3C 的得分散点图，补充图 S7 显示了其他三种方法的结果。

通常，每种方法与游戏的组合都存在一个学习率范围，在该范围内大多数初始化都能取得良好性能，表明所有方法对学习率和随机初始化的选择都较为鲁棒。此外，在良好学习率范围内几乎没有得分为 0 的点，说明这些方法在训练过程中不会崩溃或发散，具有良好的稳定性。

6 结论与讨论

我们提出了四种标准的强化学习算法的异步变体，并展示了它们能够在多种领域中稳定地训练神经网络控制器。我们的结果显示，在我们的框架中，无论是基于价值的方法还是基于策略的方法，无论是离轨还是在线策略方法，无论是离散还是连续领域，都可以通过强化学习实现稳定的训练。当在 Atari 领域使用 16 个 CPU 核心进行训练时，我们提出的异步算法比在 Nvidia K40 GPU 上训练的 DQN 更快，其中 A3C 在一半的训练时间内就超越了当前的最先进水平。此外，我们还展示了 A3C 在一系列连续控制任务以及一项仅通过视觉输入在随机 3D 迷宫中导航的新任务中同样表现出色。

我们的主要发现之一是，使用并行 actor-learner 更新共享模型对三种基于价值的方法（Q-learning、Sarsa、n 步 Q-learning）的学习过程具有稳定化作用。这表明，即使不使用经验回放，也可以实现稳定的在线 Q-learning，经验回放在 DQN 中用于此目的。然而，这并不意味着经验回放没有用。将经验回放整合到异步强化学习框架中，可以通过重用旧数据显著提高这些方法的数据效率，从而在像 TORCS 这样与环境交互成本较高的领域中进一步加快训练速度。

将其他现有的强化学习方法或近年来深度强化学习的进展与我们的异步框架结合起来，为我们提出的方法提供了许多立即改进的可能性。虽然我们的 n 步方法通过使用修正的 n 步回报直接作为目标，采用了前向视角（Sutton & Barto, 1998），但更常见的是使用后向视角，通过资格迹（eligibility traces）（Watkins, 1989; Sutton & Barto, 1998; Peng & Williams, 1996）隐式地组合不同的回报。异步优势 actor-critic 方法可以通过使用其他优势函数估计方法，如广义优势估计（Schulman et al., 2015b）来潜在改进。我们研究的所有基于价值的方法都可以从不同的减少 Q 值过估计偏差的方法中受益（Van Hasselt et al., 2015; Bellemare et al., 2016）。另一个更

投机的方向是尝试将近期关于真实在线时差分方法（van Seijen et al., 2015）的工作与非线性函数逼近结合起来。

除了这些算法改进之外，神经网络架构本身也存在许多互补性的改进空间。例如，对抗架构（Wang et al., 2015）通过在网络中包含单独的状态价值和优势流，已被证明可以产生更准确的 Q 值估计。空间 Softmax（Levine et al., 2015）可以改善网络表示特征的能力，从而改善基于价值和基于策略的方法。

总之，我们相信异步强化学习框架为深度强化学习的研究与应用提供了一个更高效、更通用、更可扩展的基础。我们希望这项工作能够激发更多关于并行强化学习、稳定训练机制以及跨领域通用智能体设计的研究。

致谢

我们感谢 Thomas Degris, Remi Munos, Marc Lanctot, Sasha Vezhnevets 和 Joseph Modayil 对本文的许多有益讨论、建议和评论。我们也感谢 DeepMind 评估团队为评估智能体设置的环境。

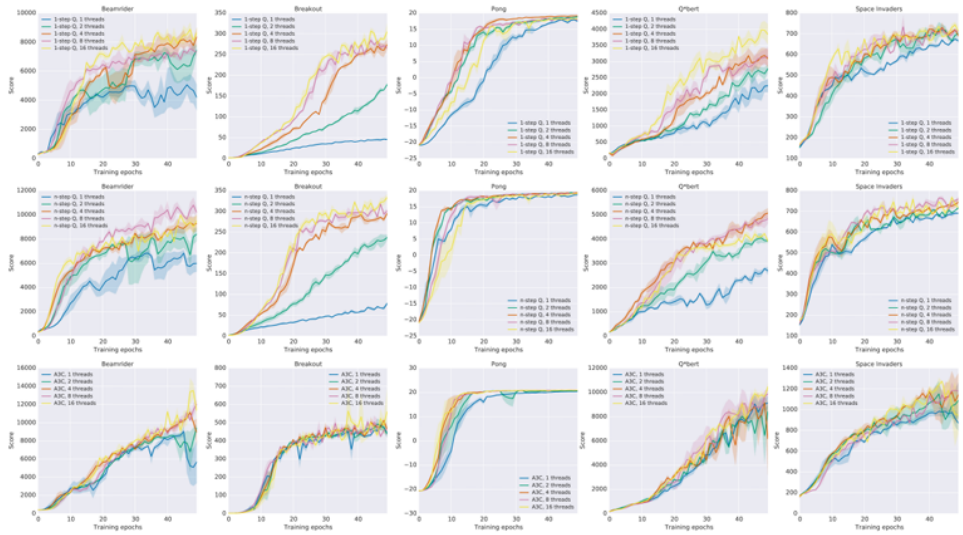


图 0.39　不同并行 worker 数量下，三种异步方法在 5 款 Atari 游戏上的数据效率对比。横轴为总训练帧数（4 M 帧/epoch），纵轴为平均得分。一步方法（1-step Q 与 Sarsa）随 worker 增加表现出更高的数据效率，说明并行更新有助于减小偏差。Sarsa 的结果见补充图 S5。

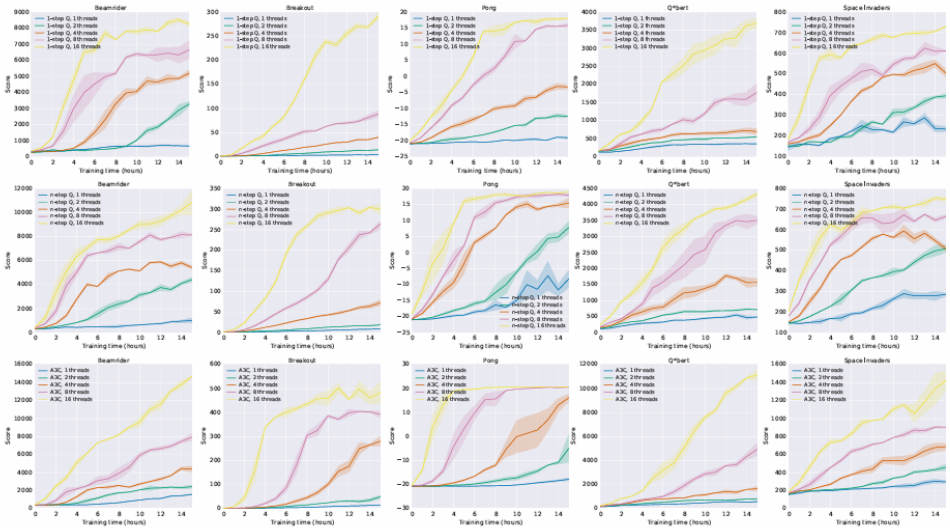


图 0.40　不同并行 worker 数量下的墙钟时间加速对比（5 款游戏）。横轴为训练小时数，纵轴为平均得分。所有异步方法均随 worker 数量增加而显著缩短训练时间，验证框架的可扩展性。Sarsa 的结果见补充图 S6。

参考文献

[1] M. Riedmiller, “Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method,” in *Machine Learning: ECML 2005*. Springer, 2005, pp. 317–328.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with deep reinforcement learning,” in *NIPS Deep Learning Workshop*, 2013.

[3] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” arXiv preprint arXiv:1509.06461, 2015.

[5] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” in *ICML*, 2015.

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” in *ICLR*, 2016.

[7] A. Nair *et al.*, “Massively parallel methods for deep reinforcement learning,” in *ICML Deep Learning Workshop*, 2015.

[8] Y. Li and D. Schuurmans, “MapReduce for parallel reinforcement learning,” in *EWRL*. Springer, 2011, pp. 309–320.

[9] M. Grounds and D. Kudenko, “Parallel reinforcement learning with linear function approximation,” in *Adaptive and Learning Agents and Multi-Agent Systems*. Springer, 2008, pp. 60–74.

[10] J. N. Tsitsiklis, “Asynchronous stochastic approximation and Q-learning,” *Machine Learning*, vol. 16, no. 3, pp. 185–202, 1994.

[11] D. P. Bertsekas, “Distributed dynamic programming,” *IEEE Transactions on Automatic Control*, vol. 27, no. 3, pp. 610–616, 1982.

[12] M. Tomassini, *Parallel and distributed evolutionary algorithms: a review*. Springer, 1999.

[13] J. Koutník, J. Schmidhuber, and F. Gomez, “Evolving deep unsupervised convolutional networks for vision-based reinforcement learning,” in *GECCO*, 2014, pp. 541–548.

[14] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[16] J. Peng and R. J. Williams, “Incremental multi-step Q-learning,” *Machine Learning*, vol. 22, no. 1–3, pp. 283–290, 1996.

[17] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *ICLR*, 2016.

[18] H. van Seijen *et al.*, "True online temporal-difference learning," arXiv preprint arXiv:1512.04087, 2015.

[19] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *ICML*, 2016.

[20] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," in *ICLR*, 2016.

[21] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2012.

[22] B. Wymann *et al.*, "TORCS: The open racing car simulator," `http://www.torcs.org`, 2013.

[23] E. Todorov, "MuJoCo: Modeling, simulation and visualization of multi-joint dynamics with contact," in *Multi-Joint Dynamics with Contact*, 2015.

[24] K. Chavez, H. Y. Ong, and A. Hong, "Distributed deep Q-learning," Stanford University Technical Report, 2015.

[25] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3–4, pp. 229–256, 1992.

[26] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *American Control Conference*, 2012, pp. 2177–2182.

[27] B. Recht, C. Ré, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *NIPS*, 2011, pp. 693–701.

《异步深度强化学习方法》补充材料[1,2]

1 优化细节

我们研究了两种不同的优化算法与我们的异步框架——随机梯度下降和 RMSProp。我们的这些算法的实现没有使用任何锁，以便在使用大量线程时最大化吞吐量。

动量随机梯度下降

在异步设置中，SGD 的实现相对直接且研究较为充分 [Recht et al., 2011]。设 θ 为所有线程共享的参数向量，$\Delta\theta_i$ 为线程 i 计算的关于参数 θ 的损失梯度。每个线程 i 独立应用标准的动量 SGD 更新：

$$m_i = \alpha m_i + (1-\alpha)\Delta\theta_i,$$

随后更新参数：

$$\theta \leftarrow \theta - \eta m_i,$$

其中，学习率为 η，动量为 α，且不使用任何锁。注意，在这种设置中，每个线程维护其自己的独立梯度和动量向量。

RMSProp

尽管 RMSProp [Tieleman and Hinton, 2012] 在深度学习文献中被广泛使用，但在异步优化设置中尚未得到广泛研究。标准的非中心化 RMSProp 更新公式为：

$$g = \alpha g + (1-\alpha)\Delta\theta^2,$$
$$\theta \leftarrow \theta - \eta\frac{\Delta\theta}{\sqrt{g+\epsilon}},$$

其中所有操作均为逐元素进行。为了在异步优化设置中应用 RMSProp，必须决定自身和平方梯度的移动平均值 g 是共享的还是每个线程独立的。我们实验了算法的两个版本。在一个版本中，我们称之为 RMSProp，每个线程维护自己的 g，如公式 S1 所示。在另一个版本中，我们称之为共享 RMSProp，向量 g 在各线程间共享，并且异步更新且不使用锁。在线程间共享统计信息还可以通过减少每个线程的参

[1]原文：Supplementary Material for ”Asynchronous Methods for Deep Reinforcement Learning”

[2]译者：陈子涵。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

数向量副本数量来降低内存需求。

我们比较了这三种异步优化算法对不同学习率和随机网络初始化的敏感性。图 S1 展示了使用两种不同的强化学习方法（异步 n 步 Q 学习和异步优势演员-评论家）在四个不同游戏（Breakout、Beamrider、Seaquest 和 Space Invaders）上的方法比较。每条曲线显示了 50 次实验的得分，这些实验对应于 50 个不同的随机学习率和初始化。x 轴显示按最终平均得分降序排列的模型排名，y 轴显示相应模型获得的最终平均得分。在这种表示中，表现更好的算法将在 y 轴上获得更高的最大奖励，而最稳健的算法将具有最接近水平的斜率，从而最大化曲线下面积。共享统计信息的 RMSProp 比每个线程具有独立统计信息的 RMSProp 更稳健，而后者又比动量 SGD 更稳健。

2 实验设置

在 Atari 游戏子集（图 1、3、4 和表 2）以及 TORCS 实验（图 S2）中进行的实验使用了以下设置。每个实验使用 16 个 actor-learner 线程在单台机器上运行，且不使用 GPU。所有方法在每 5 次动作后进行更新（$t_{max} = 5$ 和 $I_{Update} = 5$），并使用共享 RMSProp 进行优化。三种异步基于价值方法使用每 40000 帧更新一次的共享目标网络。Atari 实验使用了与 Mnih 等人 [2015] 相同的输入预处理，并且动作重复为 4。代理使用了 Mnih 等人 [2013] 的网络架构。网络包含一个具有 16 个大小为 8×8、步长为 4 的卷积层，随后是一个具有 32 个大小为 4×4、步长为 2 的卷积层，再随后是一个具有 256 个隐藏单元的全连接层。所有三个隐藏层后均跟随一个 ReLU 非线性激活函数。基于价值方法对于每个动作都有一个线性输出单元，代表动作值。演员-评论家模型具有两组输出——一个 softmax 输出，每个动作有一个条目，代表选择该动作的概率，以及一个线性输出，代表价值函数。所有实验均使用折扣因子 $\gamma = 0.99$ 和 RMSProp 衰减因子 $\alpha = 0.99$。

基于价值方法从取三个值 $\epsilon_1, \epsilon_2, \epsilon_3$ 的概率分布中采样探索率 ϵ，其概率分别为 0.4、0.3、0.3。在前四百万帧中，$\epsilon_1, \epsilon_2, \epsilon_3$ 的值分别从 1 逐渐降低到 0.1、0.01、0.5。优势演员-评论家使用了权重为 $\beta = 0.01$ 的熵正则化，用于所有 Atari 和 TORCS 实验。我们为五个 Atari 游戏和每个 TORCS 级别进行了 50 次实验，每次实验使用不同的随机初始化和初始学习率。初始学习率从 LogUniform$(10^{-}4, 10^{-}2)$ 分布中采样，并在训练过程中逐渐降低至 0。注意，在与先前工作进行比较时（表 1 和 S1），我们遵循标准评估协议并使用固定的超参数。

3 使用 MuJoCo 物理模拟器进行连续动作控制

为了将异步优势演员-评论家算法应用于 MuJoCo 任务，所需的设置几乎与离散动作域中使用的设置相同，因此这里我们仅列举连续动作域所需的差异。许多任务的基本元素（即物理模型和任务目标）与 [Lillicrap et al., 2015] 中检查的任务几乎相同。然而，由于 MuJoCo 的开发者改变了接触模型，因此对于大多数任务而言，奖励以及因此产生的性能并不具有可比性。

在所有领域中，我们尝试使用物理状态作为输入来学习任务。物理状态包括关节位置和速度，以及如果任务需要目标，则包括目标位置。此外，对于三个任务（摆、二维点质量和平行夹持器），我们还检查了直接从 RGB 像素输入进行训练的情况。在低维物理状态情况下，输入通过一个具有 200 个 ReLU 单元的隐藏层映射到隐藏状态。在使用像素的情况下，输入通过两层空间卷积，且没有任何非线性或池化操作。在任何情况下，编码器层的输出都被送入一个由 128 个 LSTM 单元组成的单层。架构中最重要的不同之处在于策略网络的输出层。与离散动作域中动作输出为 Softmax 不同，这里的策略网络的两个输出是两个实数向量，我们将它们视为具有球形协方差的多维正态分布的均值向量 μ 和方差 σ^2。在执行动作时，输入通过模型传递到输出层，我们从由 μ 和 σ^2 确定的正态分布中进行采样。在实践中，μ 由一个线性层建模，σ^2 由 SoftPlus 操作 $\log(1+\exp(x))$ 作为激活函数，作为线性层输出的函数进行计算。在我们对连续控制问题的实验中，策略网络和价值网络的参数不共享，尽管这一细节可能并不关键。最后，由于幕通常最多只有几百个时间步长，因此我们在策略或价值函数更新中没有使用任何引导，并且将每个幕批处理为一个单一更新。

与离散动作情况一样，我们包括了一个熵成本以鼓励探索。在连续情况下，我们使用了由演员网络输出定义的正态分布的微分熵的成本，$-\frac{1}{2}(\log(2\pi\sigma^2)+1)$，我们在所有检查的任务中为这个成本使用了 10^{-4} 的常数乘数。异步优势演员-评论家算法为所有领域找到了解决方案。图 S4 显示了与墙钟时间的学习曲线，并证明了大多数领域可以从状态在几个小时内解决。所有实验，包括从像素基础观察中进行的实验，都在 CPU 上运行。即使在直接从像素输入解决领域的情况下，我们也发现可以在 24 小时内可靠地发现解决方案。图 S3 显示了顶部分数与采样学习率的散点图。在大多数领域中，有一系列学习率可以一致地在任务上获得良好的性能。

算法

异步 n 步 Q 学习

算法 5 异步 n 步 Q 学习 - 每个 actor-learner 线程的伪代码

假设全局共享参数向量 θ
假设全局共享目标参数向量 θ^-
假设全局共享计数器 $T = 0$
初始化线程步数计数器 $t \leftarrow 1$
初始化目标网络参数 $\theta^- \leftarrow \theta$
初始化线程特定参数 $\theta' \leftarrow \theta$
初始化网络梯度 $d\theta \leftarrow 0$
repeat
    清空梯度 $d\theta \leftarrow 0$
    同步线程特定参数 $\theta' \leftarrow \theta$
    $t_{\text{start}} \leftarrow t$
    获取状态 s_t
    repeat
        根据基于 ϵ-贪婪策略的 $Q(s_t, a; \theta')$ 采取动作 a_t
        接收奖励 r_t 和新状态 s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until 终端状态 s_t 或 $t - t_{\text{start}} == t_{\text{max}}$
    $R \leftarrow \begin{cases} 0 & \text{对于终端状态} s_t \\ \max_a Q(s_t, a; \theta^-) & \text{对于非终端状态} s_t \end{cases}$
    for $i \in \{t-1, \ldots, t_{\text{start}}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        累积关于 θ' 的梯度: $d\theta \leftarrow d\theta + \frac{\partial (R - Q(s_i, a_i; \theta'))^2}{\partial \theta'}$
    end for
    使用 $d\theta$ 异步更新 θ
    if $T \mod I_{\text{target}} == 0$ **then**
        $\theta^- \leftarrow \theta$
    end if
until $T > T_{\text{max}}$

异步优势演员-评论家

算法 6 异步优势演员-评论家 - 每个 actor-learner 线程的伪代码

假设全局共享参数向量 θ 和 θ_v 以及全局共享计数器 $T = 0$
假设线程特定参数向量 θ' 和 θ'_v
初始化线程步数计数器 $t \leftarrow 1$
repeat
    重置梯度: $d\theta \leftarrow 0$ 和 $d\theta_v \leftarrow 0$
    同步线程特定参数 $\theta' \leftarrow \theta$ 和 $\theta'_v \leftarrow \theta_v$
    $t_{\text{start}} \leftarrow t$
    获取状态 s_t
    repeat
        根据策略 $\pi(a_t|s_t;\theta')$ 执行动作 a_t
        接收奖励 r_t 和新状态 s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until 终端状态 s_t 或 $t - t_{\text{start}} == t_{\text{max}}$
    $R \leftarrow \begin{cases} 0 & \text{对于终端状态} s_t \\ V(s_t, \theta'_v) & \text{对于非终端状态} s_t \end{cases}$
    for $i \in \{t-1, \ldots, t_{\text{start}}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        累积关于 θ' 的梯度: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        累积关于 θ'_v 的梯度: $d\theta_v \leftarrow d\theta_v + \frac{\partial (R - V(s_i;\theta'_v))^2}{\partial \theta'_v}$
    end for
    使用 $d\theta$ 异步更新 θ，使用 $d\theta_v$ 异步更新 θ_v
until $T > T_{\text{max}}$

结果与图表

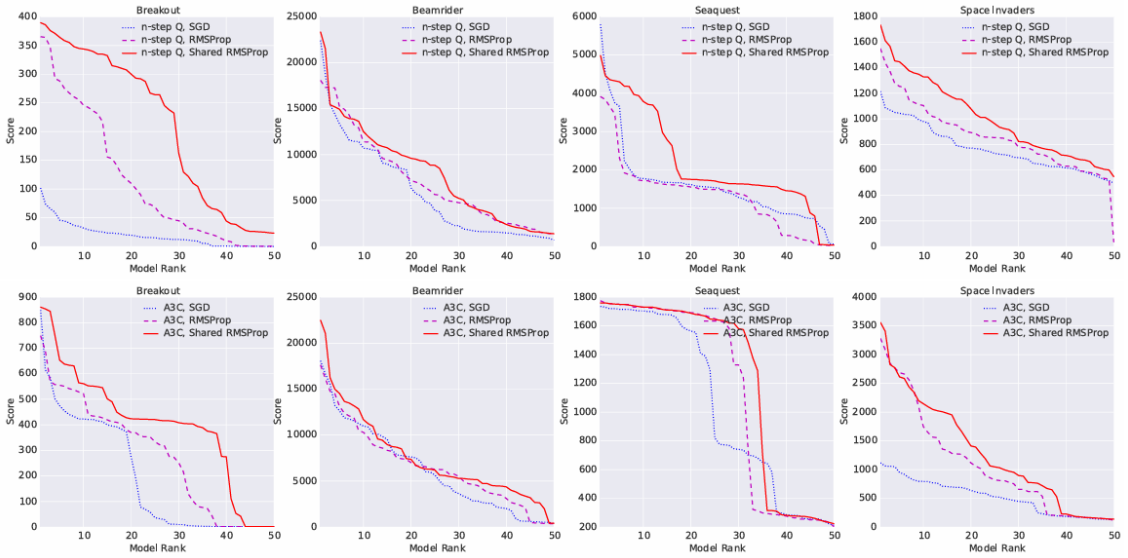


图 0.41　三种不同的优化方法（动量 SGD、RMSProp、共享 RMSProp）在四种不同的 Atari 游戏（Breakout、Beamrider、Seaquest 和 Space Invaders）上的比较。每条曲线显示了 50 次实验的最终得分，这些实验覆盖了 50 个随机初始化和学习率的搜索。顶部行显示了使用异步 n 步 Q 算法的结果，底部行显示了使用异步优势演员-评论家的结果。每个单独的图表显示了一个游戏的三个不同优化方法的结果。共享 RMSProp 比动量 SGD 和不共享的 RMSProp 对不同的学习率和随机初始化更具鲁棒性。

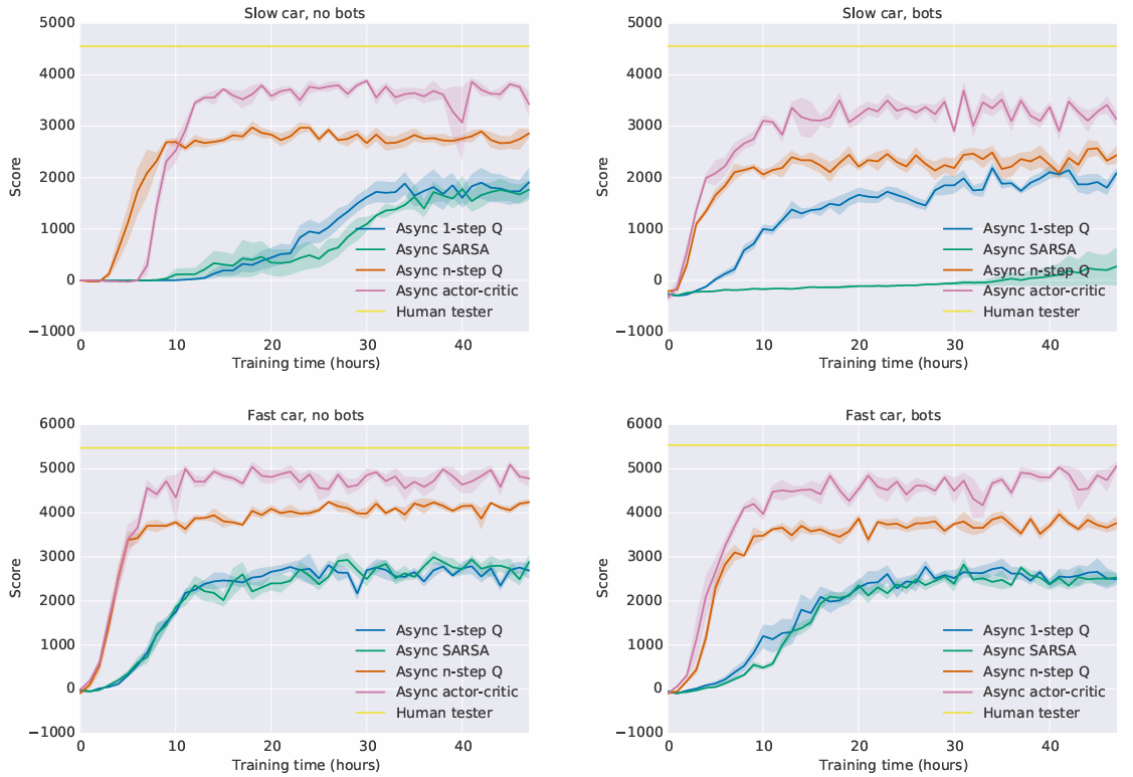


图 0.42　在 TORCS 汽车竞速模拟器上对算法进行比较。显示了四种不同的配置，包括车速和对手存在与否。在每个图表中，所有四种算法（一步 Q 学习、一步 Sarsa、n 步 Q 学习和优势演员-评论家）在训练时间（以墙上时钟小时计）与得分之间的比较。多步算法比一步算法更快地获得更好的策略。这些曲线显示了从 50 次实验中选出的 5 次最佳运行的平均值，学习率从 LogUniform($10^{-}4, 10^{-}2$)

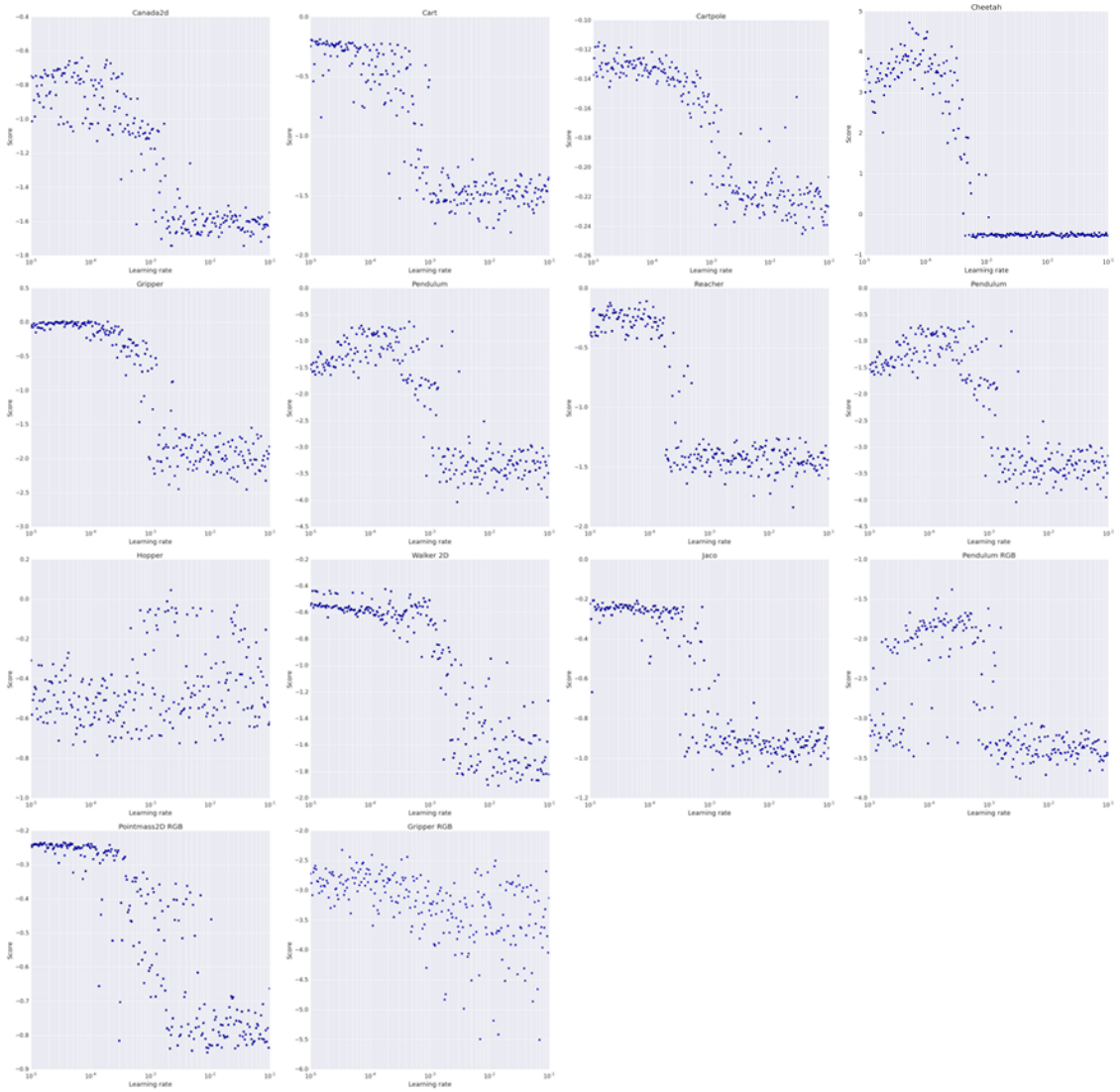


图 0.43　Mujoco 连续动作领域的性能。展示了从 LogUniform$(10^5, 10^1)$ 采样的学习率下获得的最佳得分的散点图。对于几乎所有的任务，都存在广泛的学习率范围，这些学习率能够在任务中带来良好的性能表现。

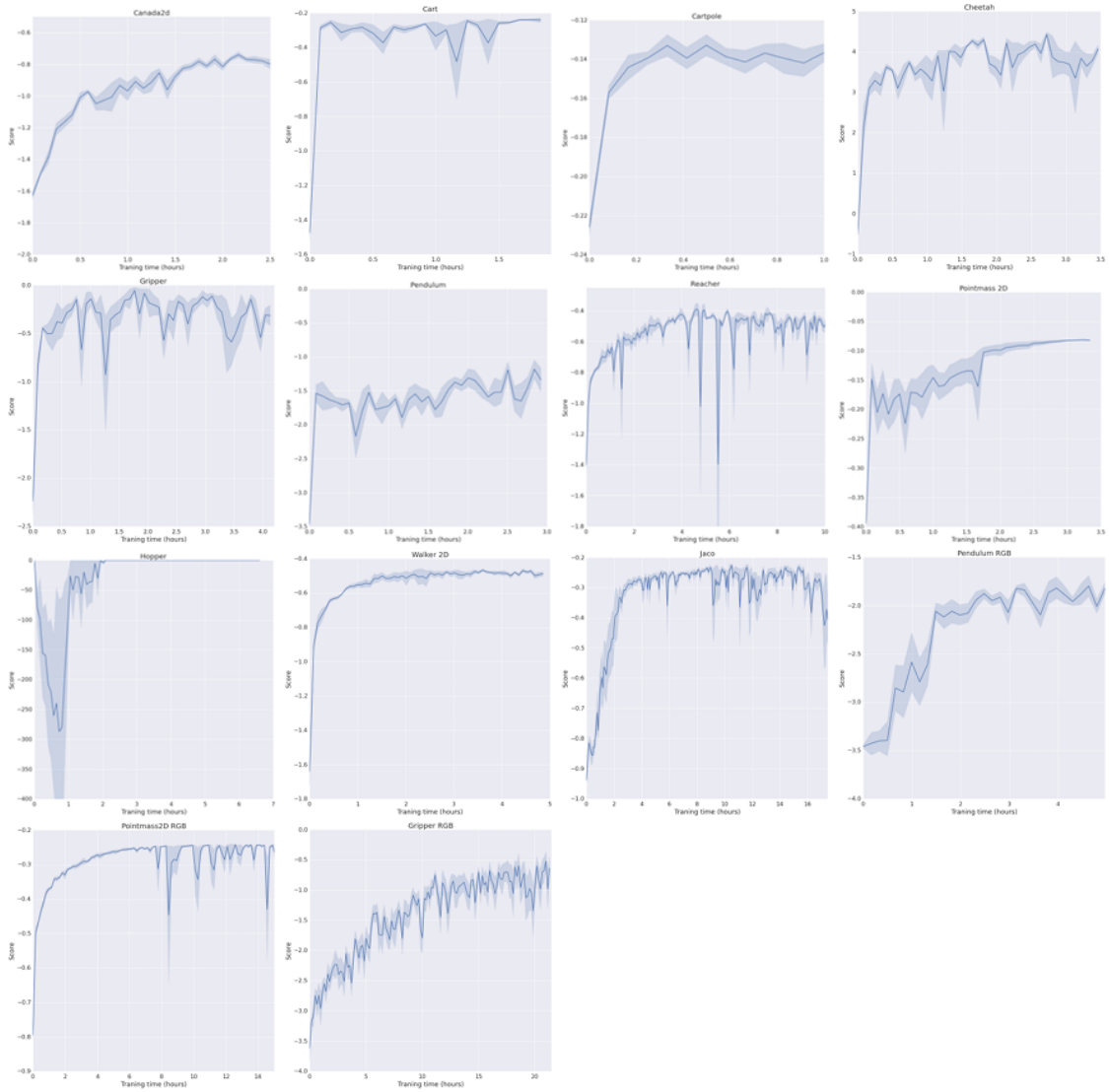


图 0.44　MuJoCo 领域的每幕得分与墙钟时间的图表。每个图表显示了前 5 次实验的误差条。

图 0.45　在五款 Atari 游戏中，不同数量的 actor-learner 进行一步 Sarsa 数据效率比较。x 轴显示的是训练周期总数，其中每个周期对应四百万帧（所有线程）。y 轴显示平均得分。每条曲线展示了在 50 个随机学习率中选取的三个表现最佳的智能体的平均值。随着并行工作者数量的增加，Sarsa 显示出数据效率的提升。

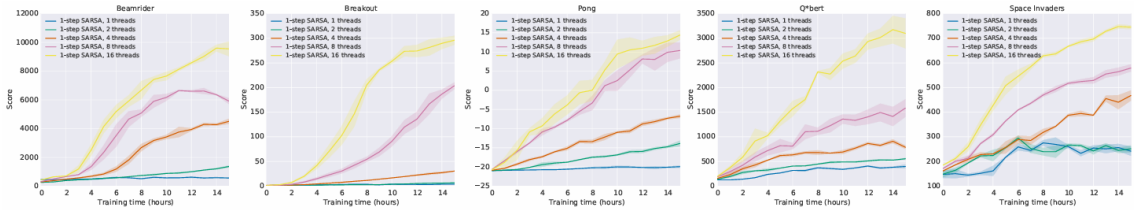


图 0.46　在五个 Atari 游戏上对不同数量的 actor-learners 进行训练速度比较。x 轴显示训练时间（小时），y 轴显示平均得分。每条曲线显示了从 50 个随机学习率搜索中选出的三个最佳性能代理的平均值。Sarsa 在使用更多并行时显示出显著的速度提升。

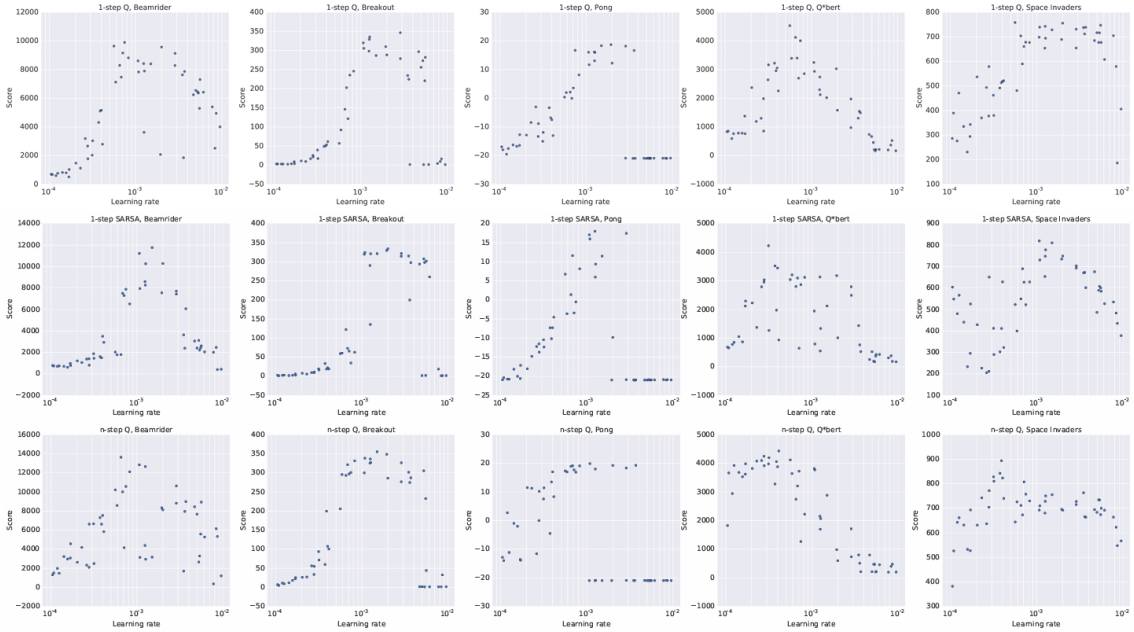


图 0.47　在五个游戏（Beamrider、Breakout、Pong、Q*bert、Space Invaders）上对一步 Q 学习、一步 Sarsa 和 n 步 Q 学习在 50 个不同的学习率和随机初始化上的分数散点图。所有算法对学习率的选择都表现出一定程度的鲁棒性。

Game	DQN	Gorila	Double	Dueling	Prioritized	A3C FF, 1 day	A3C FF	A3C LSTM
Alien	570.2	813.5	1033.4	**1486.5**	900.5	182.1	518.4	945.3
Amidar	133.4	189.2	169.1	172.7	218.4	**283.9**	263.9	173.0
Assault	3332.3	1195.8	6060.8	3994.8	7748.5	3746.1	5474.9	**14497.9**
Asterix	124.5	3324.7	16837.0	15840.0	**31907.5**	6723.0	22140.5	17244.5
Asteroids	697.1	933.6	1193.2	2035.4	1654.0	3009.4	4474.5	**5093.1**
Atlantis	76108.0	629166.5	319688.0	445360.0	593642.0	772392.0	**911091.0**	875822.0
Bank Heist	176.3	399.4	886.0	**1129.3**	816.8	946.0	970.1	932.8
Battle Zone	17560.0	19938.0	24740.0	**31320.0**	29100.0	11340.0	12950.0	20760.0
Beam Rider	8672.4	3822.1	17417.2	14591.3	**26172.7**	13235.9	22707.9	24622.2
Berzerk			1011.1	910.6	1165.6	**1433.4**	817.9	862.2
Bowling	41.2	54.0	**69.6**	65.7	65.8	36.2	35.1	41.8
Boxing	25.8	74.2	73.5	**77.3**	68.6	33.7	59.8	37.3
Breakout	303.9	313.0	368.9	411.6	371.6	551.6	681.9	**766.8**
Centipede	3773.1	**6296.9**	3853.5	4881.0	3421.9	3306.5	3755.8	1997.0
Chopper Comman	3046.0	3191.8	3495.0	3784.0	6604.0	4669.0	7021.0	**10150.0**
Crazy Climber	50992.0	65451.0	113782.0	124566.0	131086.0	101624.0	112646.0	**138518.0**
Defender			27510.0	33996.0	21093.5	36242.5	56533.0	**233021.5**
Demon Attack	12835.2	14880.1	69803.4	56322.8	73185.8	84997.5	113308.4	**115201.9**
Double Dunk	-21.6	-11.3	-0.3	-0.8	**2.7**	0.1	-0.1	0.1
Enduro	475.6	71.0	1216.6	**2077.4**	1884.4	-82.2	-82.5	-82.5
Fishing Derby	-2.3	4.6	3.2	-4.1	9.2	13.6	18.8	**22.6**
Freeway	25.8	10.2	**28.8**	0.2	27.9	0.1	0.1	0.1
Frostbite	157.4	426.6	1448.1	2332.4	**2930.2**	180.1	190.5	197.6
Gopher	2731.8	4373.0	15253.0	20051.4	**57783.8**	8442.8	10022.8	17106.8
Gravitar	216.5	**538.4**	200.5	297.0	218.0	269.5	303.5	320.0
H.E.R.O.	12952.5	8963.4	14892.5	15207.9	20506.4	28765.8	**32464.1**	28889.5
Ice Hockey	-3.8	-1.7	-2.5	-1.3	**-1.0**	-4.7	-2.8	-1.7
James Bond	348.5	444.0	573.0	835.5	**3511.5**	351.5	541.0	613.0
Kangaroo	2696.0	1431.0	**11204.0**	10334.0	10241.0	106.0	94.0	125.0
Krull	3864.0	6363.1	6796.1	8051.6	7406.5	**8066.6**	5560.0	5911.4
Kung-Fu Master	11875.0	20620.0	30207.0	24288.0	31244.0	3046.0	28819.0	**40835.0**
Montezuma's Revenge	50.0	**84.0**	42.0	22.0	13.0	53.0	67.0	41.0
Ms. Pacman	763.5	1263.0	1241.3	**2250.6**	1824.6	594.4	653.7	850.7
Name This Game	5439.9	9238.5	8960.3	11185.1	11836.1	5614.0	10476.1	**12093.7**
Phoenix			12366.5	20410.5	27430.1	28181.8	52894.1	**74786.7**
Pit Fall			-186.7	-46.9	**-14.8**	-123.0	-78.5	-135.7
Pong	16.2	16.7	**19.1**	18.8	18.9	11.4	5.6	10.7
Private Eye	298.2	**2598.6**	-575.5	292.6	179.0	194.4	206.9	421.1
Q*Bert	4589.8	7089.8	11020.8	14175.8	11277.0	13752.3	15148.8	**21307.5**
River Raid	4065.3	5310.3	10838.4	16569.4	**18184.4**	10001.2	12201.8	6591.9
Road Runner	9264.0	43079.8	43156.0	58549.0	56990.0	31769.0	34216.0	**73949.0**
Robotank	58.5	61.8	59.1	**62.0**	55.4	2.3	32.8	2.6
Seaquest	2793.9	10145.9	14498.0	37361.6	**39096.7**	2300.2	2355.4	1326.1
Skiing			-11490.4	-11928.0	**-10852.8**	-13700.0	-10911.1	-14863.8
Solaris			810.0	1768.4	**2238.2**	1884.8	1956.0	1936.4
Space Invaders	1449.7	1183.3	2628.7	5993.1	9063.0	2214.7	15730.5	**23846.0**
Star Gunner	34081.0	14919.2	58365.0	90804.0	51959.0	64393.0	138218.0	**164766.0**
Surround			1.9	**4.0**	-0.9	-9.6	-9.7	-8.3
Tennis	-2.3	-0.7	-7.8	**4.4**	-2.0	-10.2	-6.3	-6.4
Time Pilot	5640.0	8267.8	6608.0	6601.0	7448.0	5825.0	12679.0	**27202.0**
Tutankham	32.4	118.5	92.2	48.0	33.6	26.1	**156.3**	144.2
Up and Down	3311.3	8747.7	19086.9	24759.2	29443.7	54525.4	74705.7	**105728.7**
Venture	54.0	**523.4**	21.0	200.0	244.0	19.0	23.0	25.0
Video Pinball	20228.1	112093.4	367823.7	110976.2	374886.9	185852.6	331628.1	**470310.5**
Wizard of Wor	246.0	10431.0	6201.0	7054.0	7451.0	5278.0	17244.0	**18082.0**
Yars Revenge			6270.6	**25976.5**	5965.1	7270.8	7157.5	5615.5
Zaxxon	831.0	6159.4	8593.0	10164.0	9501.0	2659.0	**24622.0**	23519.0

图 0.48　人类起始条件（30 分钟模拟器时间）的原始得分。DQN 得分取自 Nair 等人 [2015]。双 DQN 分数取自 Van Hasselt 等人 [2015]。对抗 DQN 分数取自 Wang 等人 [2015]，优先级化得分取自 Schaul 等人 [2015]。

参考文献

[1] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, 和 Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, 和 Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529 – 533, 2015. URL `http://dx.doi.org/10.1038/nature14236`.

[3] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, 和 David Silver. Massively parallel methods for deep reinforcement learning. *In ICML Deep Learning Workshop*, 2015.

[4] Benjamin Recht, Christopher Re, Stephen Wright, 和 Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. *In Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[5] Tom Schaul, John Quan, Ioannis Antonoglou, 和 David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[6] Tijmen Tieleman 和 Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

[7] Hado Van Hasselt, Arthur Guez, 和 David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[8] Z. Wang, N. de Freitas, 和 M. Lanctot. Dueling network architectures for deep reinforcement learning. *ArXiv e-prints*, November 2015.

信赖域策略优化 [1,2]

摘要

我们提出一种保证单调改进的策略优化迭代方法。通过对理论推导过程进行若干近似处理，开发出名为” 置信域策略优化”（Trust Region Policy Optimization，TRPO）的实用算法。该算法与自然策略梯度方法相似，能有效优化神经网络等大型非线性策略。实验表明，该算法在多种任务中均表现优异：包括仿真机器人游泳、跳跃、步态学习等运动控制任务，以及以屏幕图像为输入的 Atari 游戏任务。尽管存在偏离理论推导的近似处理，但 TRPO 在超参数调试极少的条件下仍能保持单调改进的特性。

1. 引言

大多数策略优化算法可以分为三大类：(1) 策略迭代方法，它在估计当前策略下的价值函数和改进策略之间交替进行 Bertsekas2005；(2) 策略梯度方法，它使用从样本轨迹获得的期望回报（总奖励）梯度的估计器 Peters2008a（并且，正如我们后面讨论的，与策略迭代有密切连接）；以及 (3) 无导数优化方法，例如交叉熵方法（CEM）和协方差矩阵适应（CMA），它们将回报视为关于策略参数的黑盒函数进行优化 Szita2006。

通用的无导数随机优化方法如 CEM 和 CMA 在许多问题上更受青睐，因为它们取得了良好的结果，同时简单易懂和实现。例如，虽然俄罗斯方块是近似动态规划（ADP）方法的经典基准问题，但随机优化方法在这个任务上难以被击败 Gabillon2013。对于连续控制问题，像 CMA 这样的方法在提供手工设计的低维参数化策略类时，已经成功学习了用于挑战性任务如运动控制的策略 Wampler2009。ADP 和基于梯度的方法无法一致地击败无梯度随机搜索是不令人满意的，因为基于梯度的优化算法比无梯度方法享有更好的样本复杂性保证 Nemirovski2005。连续的基于梯度的优化在监督学习任务中学习具有大量参数的函数逼近器方面非常成功，将其成功扩展到强化学习将允许高效训练复杂和强大的策略。

在本文中，我们首先证明了最小化某个代理目标函数保证了非平凡步长的策

[1]原文: Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization[C]//International conference on machine learning. PMLR, 2015: 1889-1897.

[2]译者：王文远。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

略改进。然后我们对理论算法进行一系列近似，产生了一个实用算法，我们称之为信任域策略优化（TRPO）。我们描述了该算法的两种变体：第一，单路径方法，可应用于无模型设置；第二，vine 方法，要求系统恢复到特定状态，这通常仅在模拟中可能。这些算法是可扩展的，可以优化具有数万个参数的非线性策略，这以前对无模型策略搜索构成了重大挑战 Deisenroth2013。在我们的实验中，我们表明相同的 TRPO 方法可以学习用于游泳、跳跃和行走的复杂策略，以及直接从原始图像玩 Atari 游戏。

2. 预备知识

考虑一个无限视界折扣马尔可夫决策过程 (MDP)，其由元组 $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$ 定义，其中 $\mathcal{S}$ 是一个有限的状态集合，$\mathcal{A}$ 是一个有限的动作集合，$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ 是转移概率分布，$r : \mathcal{S} \to \mathbb{R}$ 是奖励函数，$\rho_0 : \mathcal{S} \to \mathbb{R}$ 是初始状态 s_0 的分布，$\gamma \in (0, 1)$ 是折扣因子。

令 π 表示一个随机策略 $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$，并令 $\eta(\pi)$ 表示其期望折扣回报：

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \text{其中}$$
$$s_0 \sim \rho_0(s_0),\ a_t \sim \pi(a_t|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t).$$

我们将使用以下关于状态-动作价值函数 Q_π、状态价值函数 V_π 和优势函数 A_π 的标准定义：

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right],$$
$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right],$$
$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s), \quad \text{其中}$$
$$a_t \sim \pi(a_t|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t) \text{ 对于} t \geq 0.$$

以下有用的恒等式用关于策略 π 的优势函数，以时间步累积的形式表达了另一个策略 $\tilde{\pi}$ 的期望回报（证明参见[2] 或附录 A）：

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right] \tag{1}$$

其中符号 $\mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}}[\ldots]$ 表示动作根据 $a_t \sim \tilde{\pi}(\cdot|s_t)$ 采样。

令 ρ_π 表示（未归一化的）折扣访问频率：

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \dots,$$

其中 $s_0 \sim \rho_0$ 且动作根据 π 选择。我们可以用对状态的求和来代替对时间步的求和，从而重写方程 (1)：

$$\begin{aligned}\eta(\tilde{\pi}) &= \eta(\pi) + \sum_{t=0}^{\infty} \sum_{s} P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s)\gamma^t A_\pi(s,a) \\ &= \eta(\pi) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s) A_\pi(s,a) \\ &= \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A_\pi(s,a). \end{aligned} \tag{2}$$

方程 (2) 表明，任何策略更新 $\pi \to \tilde{\pi}$，如果在每个状态 s 都具有非负的期望优势，即 $\sum_a \tilde{\pi}(a|s)A_\pi(s,a) \geq 0$，则保证能提高策略性能 η，或者在期望优势处处为零的情况下保持不变。这蕴含了精确策略迭代所执行更新的经典结果，即如果至少存在一个状态动作对有正的优势函数值和非零的状态访问概率，使用确定性策略 $\tilde{\pi}(s) = \arg\max_a A_\pi(s,a)$ 能够改进策略, 否则说明算法收敛到了最优策略。然而，在近似设定中，由于估计误差和近似误差的存在，通常不可避免地会出现某些状态的期望优势为负的情况，即 $\sum_a \tilde{\pi}(a|s)A_\pi(s,a) < 0$。$\rho_{\tilde{\pi}}(s)$ 对 $\tilde{\pi}$ 的复杂依赖关系使得方程 (2) 难以直接优化。因此，我们引入以下关于 η 的局部近似：

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_\pi(s) \sum_{a} \tilde{\pi}(a|s) A_\pi(s,a). \tag{3}$$

需要注意的是，L_π 使用的是原策略的状态访问频率 ρ_π 而非 $\rho_{\tilde{\pi}}$，即忽略了策略改变所导致的访问频率的变化。然而，如果我们有一个参数化策略 π_θ，其中 $\pi_\theta(a|s)$ 是参数向量 θ 的可微函数，那么 L_π 与 η 的一阶特性是匹配的（参见[2]）。也就是说，对于任何参数值 θ_0，有：

$$\begin{aligned} L_{\pi_{\theta_0}}(\pi_{\theta_0}) &= \eta(\pi_{\theta_0}), \\ \nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)\Big|_{\theta=\theta_0} &= \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_0}. \end{aligned} \tag{4}$$

方程 (4) 表明，一个能改进 $L_{\pi_{\theta_{old}}}$ 的、足够小的步长更新 $\pi_{\theta_0} \to \tilde{\pi}$，也将会改进 η，但这并未为我们提供关于步长应取多大的任何指导。

为了解决这个问题，Kakade2002 提出了一种称为保守策略迭代的策略更新方案，并为其提供了明确的关于 η 改进的下界。为了定义保守策略迭代更新，令 π_{old}

表示当前策略，并令 $\pi' = \arg\max_{\pi'} L_{\pi_{\text{old}}}(\pi')$。新策略 π_{new} 被定义为以下混合策略：

$$\pi_{\text{new}}(a|s) = (1-\alpha)\pi_{\text{old}}(a|s) + \alpha\pi'(a|s). \tag{5}$$

Kakade 和 Langford 推导出了如下下界：

$$\eta(\pi_{\text{new}}) \geq L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$
$$\text{其中}\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}\left[A_\pi(s,a)\right]\right|. \tag{6}$$

（我们对其进行了修改，使其稍弱但更简单。）然而需要注意的是，目前这个下界仅适用于由方程 (5) 生成的混合策略。这个策略类在实践中难以处理且限制性强，因此需要一个能适用于所有通用随机策略类的策略更新方案。

3. 广义随机策略的单调性改进保证

不等式 (6) 适用于保守策略迭代，表明如果一次策略更新能改善不等式右边，那么保证改善真正的表现 η 我们主要的理论结果是，通过将 α 替换为 π 与 $\tilde{\pi}$ 之间的距离度量，并相应地修改常数 ϵ，不等式 (6) 中的策略改进下界可以从混合策略推广到一般的随机策略。由于混合策略在实践中很少使用，这一结果对于将改进保证扩展到实际问题至关重要。我们所使用的具体距离度量是总变分散度（total variation divergence），对于离散概率分布 p, q，[1] 其定义为 $D_{\text{TV}}(p\|q) = \frac{1}{2}\sum_i |p_i - q_i|$。定义 $D_{\text{TV}}^{\max}(\pi, \tilde{\pi})$ 为：

$$D_{\text{TV}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\text{TV}}(\pi(\cdot|s)\|\tilde{\pi}(\cdot|s)). \tag{7}$$

定理 1. 令 $\alpha = D_{\text{TV}}^{\max}(\pi_{\text{old}}, \pi_{\text{new}})$。则下列下界成立：

$$\eta(\pi_{\text{new}}) \geq L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$
$$\text{其中}\epsilon = \max_{s,a} |A_\pi(s,a)| \tag{8}$$

我们在附录中提供了两种证明方法。第一种证明扩展了 Kakade 和 Langford 的结果，其依据是：从总变分散度小于 α 的两个分布中采样的随机变量可以被耦合（coupled），使得它们以 $1-\alpha$ 的概率相等。第二种证明使用了摄动理论。

接下来，我们注意到总变分散度 (total variation divergence) 与 KL 散度 (Kullback-Leibler divergence) 之间存在以下关系 Pollard2000：$D_{\text{TV}}(p\|q)^2 \leq D_{\text{KL}}(p\|q)$。令 $D_{\text{KL}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\text{KL}}(\pi(\cdot|s)\|\tilde{\pi}(\cdot|s))$。那么，由定理 1 可直接得

[1] 我们的结果可以通过用积分替代求和来直接拓展到连续状态和动作空间

出以下下界：

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - CD_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}),$$
$$\text{其中} C = \frac{4\epsilon\gamma}{(1-\gamma)^2} \tag{9}$$

算法 1 描述了一个基于方程 (9) 中策略改进下界的近似策略迭代方案。请注意，目前我们假设可以精确计算优势值 A_{π}。

算法 7 保证期望回报 η 非递减的策略迭代算法

1: **初始化**策略 π_0。
2: **for** $i = 0, 1, 2, \ldots$ **直至收敛 do**
3: 计算所有状态-动作对的优势值 $A_{\pi_i}(s, a)$。
4: 求解如下约束优化问题，得到新策略：
5: $\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - CD_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$
6: **其中** $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$
7: **且** $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$
8: **end for**

由方程 (9) 可知，算法 1 保证能生成一个策略性能单调递增的序列 $\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq \cdots$。为证明这一点，令 $M_i(\pi) = L_{\pi_i}(\pi) - CD_{\mathrm{KL}}^{\max}(\pi_i, \pi)$。则有

$$\eta(\pi_{i+1}) \geq M_i(\pi_{i+1}) \quad \text{（由方程 (9)）}$$
$$\eta(\pi_i) = M_i(\pi_i), \quad \text{因此}$$
$$\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M_i(\pi_i). \tag{10}$$

因此，通过在每次迭代中最大化 M_i，我们保证真实目标 η 是非递减的。此算法是一种最小化-最大化（MM）算法[3]，这类方法中也包括期望最大化（EM）算法。在 MM 算法的术语中，M_i 是一个代理函数（surrogate function），它在 π_i 处与 η 相等，且是 η 的一个下界函数（minorizes η）。该算法也让人联想到近端梯度法和镜像下降法。我们在下一节提出的**信任域策略优化**（Trust Region Policy Optimization）是算法 1 的一个近似，它使用对 KL 散度的约束（而非惩罚项）来鲁棒地允许较大的策略更新。

4. 参数化策略的优化

在前一节中，我们在独立于策略 π 的参数化、并假设策略在所有状态都可评估的前提下，考虑了策略优化问题。现在，我们将阐述如何从这些理论基础出发，在有限样本和任意参数化的情况下，推导出一个实用算法。

由于我们考虑的是带有参数向量 θ 的参数化策略 $\pi_{\theta}(a|s)$，我们将重载之前的

符号，使用关于 θ 的函数而非 π，例如 $\eta(\theta) := \eta(\pi_\theta)$，$L_\theta(\tilde{\theta}) := L_{\pi_\theta}(\pi_{\tilde{\theta}})$，以及 $D_{\mathrm{KL}}(\theta\|\tilde{\theta}) := D_{\mathrm{KL}}(\pi_\theta\|\pi_{\tilde{\theta}})$。我们将使用 θ_{old} 表示我们希望改进的先前策略参数。

前一节表明 $\eta(\theta) \geq L_{\theta_{\text{old}}}(\theta) - CD^{\max}_{\mathrm{KL}}(\theta_{\text{old}}, \theta)$，且在 $\theta = \theta_{\text{old}}$ 时取等号。因此，通过执行以下优化，我们保证能改进真实目标 η：

$$\max_\theta \left[L_{\theta_{\text{old}}}(\theta) - CD^{\max}_{\mathrm{KL}}(\theta_{\text{old}}, \theta)\right].$$

然而在实践中，如果我们使用上述理论推荐的惩罚系数 C，步长将会非常小。一种以鲁棒方式采取更大步长的方法是使用新策略与旧策略之间 KL 散度的约束，即信任域约束：

$$\begin{aligned} \max_\theta \quad & L_{\theta_{\text{old}}}(\theta) \\ \text{subject to} \quad & D^{\max}_{\mathrm{KL}}(\theta_{\text{old}}, \theta) \leq \delta. \end{aligned} \tag{11}$$

该问题施加了一个约束，即 KL 散度在状态空间的每一点上都有界。虽然这是由理论所驱动的，但由于约束数量庞大，该问题实际上难以求解。因此，我们可以使用一种启发式近似，即考虑平均 KL 散度：

$$\overline{D}^{\rho}_{\mathrm{KL}}(\theta_1, \theta_2) := \mathbb{E}_{s\sim\rho}\left[D_{\mathrm{KL}}(\pi_{\theta_1}(\cdot|s) \parallel \pi_{\theta_2}(\cdot|s))\right].$$

因此，我们提出求解以下优化问题来生成策略更新：

$$\begin{aligned} \underset{\theta}{\text{maximize}} \ & L_{\theta_{\text{old}}}(\theta) \\ \text{subject to } & \overline{D}^{\rho_{\theta_{\text{old}}}}_{\mathrm{KL}}(\theta_{\text{old}}, \theta) \leq \delta. \end{aligned} \tag{12}$$

类似的策略更新在先前的工作中已被提出[4–6]，我们将在第 7 节和第 8 节的实验中，将我们的方法与先前的方法进行比较。我们的实验也表明，这种类型的约束更新与方程 (11) 中的最大 KL 散度约束具有相似的实证性能。

5. 目标函数与约束的基于样本估计

上一节提出了一个关于策略参数的约束优化问题（12），该问题在每次更新时优化期望总回报 η 的一个估计，并受限于策略变化的约束。本节将描述如何利用蒙特卡罗模拟来近似目标函数和约束函数。

我们旨在求解通过展开方程 (12) 中的 $L_{\theta_{\text{old}}}$ 得到的以下优化问题：

$$\begin{aligned} \underset{\theta}{\text{maximize}} \ & \sum_s \rho_{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{\text{old}}}(s,a) \\ \text{subject to } & \overline{D}^{\rho_{\theta_{\text{old}}}}_{\mathrm{KL}}(\theta_{\text{old}}, \theta) \leq \delta. \end{aligned} \tag{13}$$

我们首先用期望 $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}[\cdots]$ 替换目标函数中的 $\sum_s \rho_{\theta_{\text{old}}}(s)[\cdots]$。接着，我们将方程 (13) 中的优势值 $A_{\theta_{\text{old}}}$ 替换为 Q 值 $Q_{\theta_{\text{old}}}$，这只会使目标函数改变一个常数。最后，我们使用一个重要性采样估计器来替换对动作的求和。用 q 表示采样分布，单个状态 s_n 对损失函数的贡献为：

$$\sum_a \pi_\theta(a|s_n) A_{\theta_{\text{old}}}(s_n, a) = \mathbb{E}_{a\sim q}\left[\frac{\pi_\theta(a|s_n)}{q(a|s_n)} A_{\theta_{\text{old}}}(s_n, a)\right].$$

方程 (13) 中的优化问题完全等价于以下用期望形式书写的问题：

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\ \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{\text{old}}}(s, a)\right] \\ &\text{subject to}\ \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s) \parallel \pi_\theta(\cdot|s))\right] \le \delta. \end{aligned} \tag{14}$$

接下来要做的就是使用样本均值来替换期望，并使用经验估计来替换 Q 值。以下两小节将描述执行此估计的两种不同方案。

第一种抽样方案，我们称之为单路径方法（*single-path*），是策略梯度估计中通常使用的方案[7]，它基于对单条轨迹的抽样。第二种方案，我们称之为 **Vine 方法**（*vine*），需要先构建一个 *rollout* 集，然后从该集合中的每个状态执行多个动作。这种方法主要在策略迭代方法的背景下被探索过[8,9]。

5.1 单路径方法 (Single Path)

在此估计过程中，我们通过抽样初始状态 $s_0 \sim \rho_0$ 来收集一个状态序列，然后模拟策略 $\pi_{\theta_{\text{old}}}$ 若干时间步，以生成一条轨迹 $s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T$。因此，采样分布为 $q(a|s) = \pi_{\theta_{\text{old}}}(a|s)$。通过计算轨迹上未来奖励的折扣和，来得到每个状态-动作对 (s_t, a_t) 处的 $Q_{\theta_{\text{old}}}(s, a)$ 值。

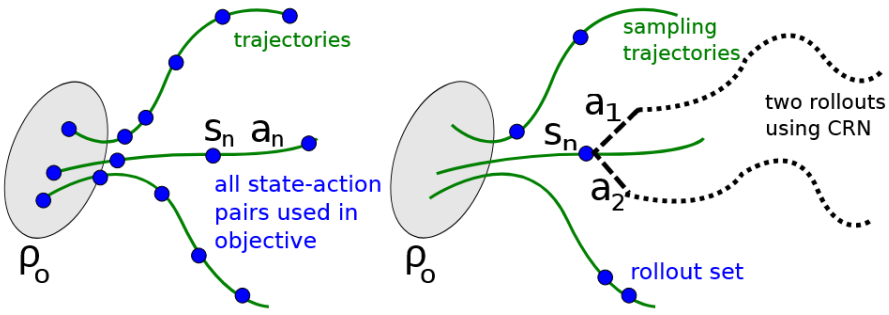


图 0.49　左图：单路径方法示意图。这里，我们通过策略模拟生成一组轨迹，并将所有状态-动作对 (s_n, a_n) 纳入目标。右图：藤蔓方法示意图。我们生成一组“主干”轨迹，然后从到达的状态子集中生成“分支”回滚。对于每个状态 s_n，我们执行多个动作（这里为 a_1 和 a_2），并在每个动作后执行回滚，使用公共随机数（CRN）来减少方差。

5.2 Vine 方法

在此估计过程中，我们首先抽样初始状态 $s_0 \sim \rho_0$ 并模拟策略 π_{θ_i} 以生成若干条轨迹。然后，我们沿着这些轨迹选择一个包含 N 个状态的子集，记为 $s_1, s_2, \ldots, s_N$，我们称之为“*rollout* 集”。对于 *rollout* 集中的每个状态 s_n，我们根据分布 $a_{n,k} \sim q(\cdot|s_n)$ 抽样 K 个动作。任何其支撑集包含 $\pi_{\theta_i}(\cdot|s_n)$ 支撑集的分布 $q(\cdot|s_n)$ 都将产生一个一致估计量。在实践中，我们发现 $q(\cdot|s_n) = \pi_{\theta_i}(\cdot|s_n)$ 在连续性问题（例如机器人运动）上效果很好，而均匀分布在离散任务（例如 Atari 游戏）上效果很好，因为后者有时能实现更好的探索。

对于在每个状态 s_n 处采样的每个动作 $a_{n,k}$，我们通过从状态 s_n 和动作 $a_{n,k}$ 开始执行一个 *rollout*（即一条短轨迹）来估计 $\hat{Q}_{\theta_i}(s_n, a_{n,k})$。通过在每个 K 条 *rollout* 中使用相同的随机数序列作为噪声，即使用公共随机数（common random numbers），我们可以大大减少 *rollout* 之间 Q 值差异的方差。关于 Q 值的蒙特卡罗估计的更多讨论，请参见工作[10]；关于强化学习中公共随机数的讨论，请参见工作[11]。

在小型、有限的动作空间中，我们可以为给定状态下的每个可能动作生成一个 *rollout*。单个状态 s_n 对 $L_{\theta_{\text{old}}}$ 的贡献如下：

$$L_n(\theta) = \sum_{k=1}^{K} \pi_\theta(a_k|s_n)\hat{Q}(s_n, a_k), \tag{15}$$

其中动作空间为 $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$。

在大型或连续的状态空间中，我们可以使用重要性采样来构建代理目标函数的估计量。在单个状态 s_n 处获得的 $L_{\theta_{\text{old}}}$ 的*自归一化估计量*（self-normalized estimator）[12] 为：

$$L_n(\theta) = \frac{\sum_{k=1}^{K} \frac{\pi_\theta(a_{n,k}|s_n)}{\pi_{\theta_{\text{old}}}(a_{n,k}|s_n)}\hat{Q}(s_n, a_{n,k})}{\sum_{k=1}^{K} \frac{\pi_\theta(a_{n,k}|s_n)}{\pi_{\theta_{\text{old}}}(a_{n,k}|s_n)}}, \tag{16}$$

此式假设我们从状态 s_n 执行了 K 个动作 $a_{n,1}, a_{n,2}, \ldots, a_{n,K}$。该自归一化估计量移除了为 Q 值使用基线（baseline）的需要（注意，给 Q 值加上一个常数不会改变梯度）。通过对 $s_n \sim \rho(\pi)$ 求平均，我们得到了 $L_{\theta_{\text{old}}}$ 及其梯度的估计量。

Vine 方法和单路径（single path）方法如图 1 所示。我们使用术语 vine，因为用于采样的轨迹可以比作藤蔓的茎，它们在各个点（即 rollout 集）分叉成几个短小的分枝（即 rollout 轨迹）。

与单路径方法相比，vine 方法的好处在于，在代理目标函数中相同数量的 Q 值样本下，我们对目标的局部估计具有更低的方差。也就是说，vine 方法能提供更好的优势值估计。vine 方法的缺点是我们必须为每个优势估计执行更多次模拟器

调用。此外，vine 方法要求我们从 rollout 集中的每个状态生成多条轨迹，这限制了该算法只能用于可以将系统重置到任意状态的设置。相比之下，单路径算法不需要状态重置，可以直接在物理系统上实现[5]。

6. 实用算法

这里我们基于上述思想提出两种实用的策略优化算法，它们分别使用前一小节中的单路径或 vine 采样方案。这些算法重复执行以下步骤：

1. 使用单路径（single path）或 vine 方法收集一组状态-动作对及其 Q 值的蒙特卡罗估计。
2. 通过对样本求平均，构建方程 (14) 中的估计目标函数和约束条件。
3. 近似求解该约束优化问题，以更新策略的参数向量 θ。我们使用共轭梯度算法（conjugate gradient algorithm）followed by a line search，其总计算成本仅比计算梯度本身略高。详见附录 C。

关于步骤 (3)，我们通过解析计算 KL 散度的 Hessian 矩阵来构造费舍尔信息矩阵（FIM, Fisher information matrix），而不是使用梯度的协方差矩阵。也就是说，我们将 A_{ij} 估计为：$\frac{1}{N}\sum_{n=1}^{N}\frac{\partial^2}{\partial\theta_i\partial\theta_j}D_{\mathrm{KL}}(\pi_{\theta_{\mathrm{old}}}(\cdot|s_n)\ \|\ \pi_\theta(\cdot|s_n))$，而不是 $\frac{1}{N}\sum_{n=1}^{N}\frac{\partial}{\partial\theta_i}\log\pi_\theta(a_n|s_n)\frac{\partial}{\partial\theta_j}\log\pi_\theta(a_n|s_n)$.

Fisher 信息[1] 是一种衡量可观测随机变量 X 携带关于未知参数 θ 信息量的方法，其中 X 的概率依赖于 θ。设 $f(X;\theta)$ 是给定 θ 值时 X 的概率密度函数（或概率质量函数）。它描述了在已知 θ 值的情况下，我们观测到 X 的某个特定结果的概率。

如果 f 相对于 θ 的变化尖锐峰值，则很容易从数据中指示 θ 的" 正确" 值，或者等价地说，数据 X 提供了关于参数 θ 的大量信息。如果 f 是平坦且分散的，那么需要许多 X 的样本来估计使用整个被抽样总体将获得的实际" 真实"θ 值。这表明需要研究关于 θ 的某种方差。

形式上，似然函数的自然对数关于 θ 的偏导数称为得分（score）。在某些正则性条件下，如果 θ 是真实参数（即 X 实际上服从 $f(X;\theta)$ 分布），则可以证明在真实参数值 θ 处评估的得分的期望值（一阶矩）为 0：

$$
\begin{aligned}
E\left[\frac{\partial}{\partial\theta}\log f(X;\theta)\Big|_\theta\right] &= \int_{\mathbb{R}}\frac{\frac{\partial}{\partial\theta}f(x;\theta)\big|_\theta}{f(x;\theta)}f(x;\theta)\,dx \\
&= \frac{\partial}{\partial\theta}\int_{\mathbb{R}}f(x;\theta)\,dx
\end{aligned}
$$

[1]翻译节选自 wikipedia 中 Fisher information 词条，链接：https://en.wikipedia.org/wiki/Fisher_information

$$= \frac{\partial}{\partial \theta} 1$$
$$= 0.$$

Fisher 信息定义为得分的方差：

$$\mathcal{I}(\theta) = E\left[\left(\frac{\partial}{\partial \theta} \log f(X;\theta)\right)^2 \middle| \theta\right] = \int_{\mathbb{R}} \left(\frac{\partial}{\partial \theta} \log f(x;\theta)\right)^2 f(x;\theta)\, dx,$$

注意 $\mathcal{I}(\theta) \geq 0$。携带高 Fisher 信息的随机变量意味着得分的绝对值通常很高。Fisher 信息不是特定观测值的函数，因为随机变量 X 已被平均掉。

如果 $\log f(x;\theta)$ 关于 θ 是二次可微的，并且在某些额外的正则性条件下，则 Fisher 信息也可以写为：

$$\mathcal{I}(\theta) = -E\left[\frac{\partial^2}{\partial \theta^2} \log f(X;\theta) \middle| \theta\right],$$

Fisher 信息与相对熵相关。两个分布 p 和 q 之间的相对熵，或 Kullback-Leibler 散度，可以写为：

$$KL(p:q) = \int p(x) \log \frac{p(x)}{q(x)}\, dx.$$

现在，考虑由参数 $\theta \in \Theta$ 参数化的概率分布族 $f(x;\theta)$。则该族中两个分布之间的 Kullback-Leibler 散度为：

$$D(\theta, \theta') = KL(p(\cdot;\theta) : p(\cdot;\theta')) = \int f(x;\theta) \log \frac{f(x;\theta)}{f(x;\theta')}\, dx.$$

如果 θ 固定，则同一族中两个分布之间的相对熵在 $\theta' = \theta$ 处取得最小值。对于接近 θ 的 θ'，可以将上述表达式展开为二阶级数：

$$D(\theta, \theta') = \frac{1}{2}(\theta' - \theta)^\top \left(\frac{\partial^2}{\partial \theta'_i\, \partial \theta'_j} D(\theta, \theta')\right)_{\theta'=\theta} (\theta' - \theta) + o\left((\theta' - \theta)^2\right)$$

其中的二阶导数可以写为：

$$\left(\frac{\partial^2}{\partial \theta'_i\, \partial \theta'_j} D(\theta, \theta')\right)_{\theta'=\theta} = -\int f(x;\theta) \left(\frac{\partial^2}{\partial \theta'_i\, \partial \theta'_j} \log(f(x;\theta'))\right)_{\theta'=\theta} dx = [\mathcal{I}(\theta)]_{i,j}.$$

因此，Fisher 信息表示条件分布相对于其参数的相对熵的曲率。

解析估计器在每个状态 s_n 上对动作进行积分，且不依赖于采样的动作 a_n。如附录 C 所述，这种解析估计器在大规模设置中具有计算优势，因为它无需存储稠密的 Hessian 矩阵或来自一批轨迹的所有策略梯度。实验表明，策略的改进速度与经验 FIM 相似。

让我们简要总结一下第 3 节中的理论与我们所描述的实际算法之间的关系：

- 理论证明了优化一个带有 KL 散度惩罚项的代理目标函数是合理的。然而，

过大的惩罚系数 C 会导致步长小到不可行，因此我们希望减小这个系数。根据经验，很难鲁棒地选择惩罚系数，因此我们使用一个带有参数 δ（KL 散度的边界）的硬约束来代替惩罚项。

- 对 $D_{\mathrm{KL}}^{\max}(\theta_{\mathrm{old}}, \theta)$ 的约束对于数值优化和估计来说是困难的，因此我们转而约束 $\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}}, \theta)$。
- 我们的理论忽略了优势函数的估计误差。工作[2] 在他们的推导中考虑了这种误差，并且相同的论点在本文的设定中也成立，但为了简洁起见我们将其省略了。

7. 与已有工作的联系

如第 4 节所述，我们的推导产生了一种策略更新方法，该方法与先前的一些方法相关，为许多策略更新. 自然策略梯度方法[2] 可以看作是方程 (12) 中更新的一个特殊情况，通过对 L 进行线性近似并对 $\overline{D}_{\mathrm{KL}}$ 约束进行二次近似得到，从而产生以下优化问题：

$$\begin{aligned}
&\underset{\theta}{\text{maximize}} \left[\nabla_\theta L_{\theta_{\mathrm{old}}}(\theta)|_{\theta=\theta_{\mathrm{old}}} \cdot (\theta - \theta_{\mathrm{old}}) \right] \\
&\text{subject to } \frac{1}{2}(\theta_{\mathrm{old}} - \theta)^T A(\theta_{\mathrm{old}})(\theta_{\mathrm{old}} - \theta) \le \delta, \\
&\text{其中} A(\theta_{\mathrm{old}})_{ij} = \\
&\qquad \frac{\partial}{\partial \theta_i}\frac{\partial}{\partial \theta_j} \mathbb{E}_{s\sim\rho_\pi} \left[D_{\mathrm{KL}}(\pi(\cdot|s, \theta_{\mathrm{old}}) \parallel \pi(\cdot|s, \theta)) \right] \Big|_{\theta=\theta_{\mathrm{old}}}.
\end{aligned} \tag{17}$$

KL 散度的分布空间内，找到一个向量 d，让参数 θ 移动从而最小化 Loss。[1] 可以使用拉格朗日乘子法求极值，并将 $\mathcal{L}$ 进行展开：

$$\begin{aligned}
d^* &= \arg\min_d \mathcal{L}(\theta + d) + \lambda \left(\mathrm{KL}\left[p_\theta \| p_{\theta+d}\right] - c \right) \\
&\approx \arg\min_d \mathcal{L}(\theta) + \nabla_\theta \mathcal{L}(\theta)^{\mathrm{T}} d + \frac{1}{2}\lambda d^{\mathrm{T}} \mathrm{A} d - \lambda c.
\end{aligned}$$

让对 d 的一阶导为 0，就可以求出最小值所需要的 d：

$$\begin{aligned}
0 &= \frac{\partial}{\partial d}\mathcal{L}(\theta) + \nabla_\theta \mathcal{L}(\theta)^{\mathrm{T}} d + \frac{1}{2}\lambda d^{\mathrm{T}} \mathrm{A} d - \lambda c \\
&= \nabla_\theta \mathcal{L}(\theta) + \lambda \mathrm{A} d \\
\lambda \mathrm{A} d &= -\nabla_\theta \mathcal{L}(\theta) \\
d &= -\frac{1}{\lambda}\mathrm{A}^{-1} \nabla_\theta \mathcal{L}(\theta)
\end{aligned}$$

[1] 推导参考知乎，链接：https://zhuanlan.zhihu.com/p/546885304

这说明了分布空间的最速下降方向是自然梯度方向，而不是梯度方向。文献（Natural Gradient Works Efficiently in Learning）说明了参数空间具有一定的底层结构时，函数的普通梯度并不代表它的最陡方向，而自然梯度代表它的最陡方向。因为 Fisher 信息矩阵刻画了参数变化对策略分布变化的敏感性，所以自然梯度方法还具有以下优点[1]：

1、梯度方向与参数坐标系无关了，因此不同的参数化方法不会影响梯度方向。

2、自然梯度有效控制了不同参数更新所需的步长。文献（A Natural Policy Gradient）指出 $\Delta\theta_i = \partial J/\partial\theta_i$ 是量纲不一致的，因为左边有 θ_i 的单位，而右边有 $1/\theta_i$ 的单位（并且所有的 θ_i 不一定具有相同的量纲）。比如：高斯策略中均值和方差对于策略的影响的敏感性不同，因此对不同参数使用相同的步长更新并不合理。而 Fisher 信息矩阵衡量了“每个方向对分布的敏感程度”，会自动对这些方向重新加权，让对分布变化敏感的方向更新更小。

对应的更新公式为 $\theta_{\text{new}} = \theta_{\text{old}} + \frac{1}{\lambda}A(\theta_{\text{old}})^{-1}\nabla_\theta L(\theta)|_{\theta=\theta_{\text{old}}}$，其中步长 $\frac{1}{\lambda}$ 通常被视为算法参数。这与我们的方法不同，我们的方法在每次更新时都强制执行约束。尽管这种差异可能看起来微小，但我们的实验表明，它在处理更大规模问题时能显著提高算法性能。

我们也可以通过使用 ℓ_2 约束或惩罚项来获得标准策略梯度更新：

$$\begin{aligned} &\underset{\theta}{\text{maximize}} \left[\nabla_\theta L_{\theta_{\text{old}}}(\theta)|_{\theta=\theta_{\text{old}}} \cdot (\theta - \theta_{\text{old}})\right] \\ &\text{subject to } \frac{1}{2}\|\theta - \theta_{\text{old}}\|^2 \le \delta. \end{aligned} \tag{18}$$

策略迭代更新也可以通过求解无约束问题 $\max_\pi L_{\pi_{\text{old}}}(\pi)$ 获得，其中 L 的定义见方程 (3)。

其他几种方法也采用了与方程 (12) 类似的更新方式。相对熵策略搜索（Relative entropy policy search, REPS）[6] 约束的是状态-动作边缘分布 $p(s,a)$，而 TRPO 约束的是条件分布 $p(a|s)$。与 REPS 不同，我们的方法在内部循环中不需要进行代价高昂的非线性优化。[13] 也使用了 KL 散度约束，但其目的是鼓励策略不要偏离估计的动态模型有效的区域，而我们并不尝试显式估计系统动态。工作 [14] 的工作同样基于并推广了 [2] 的结果，并且他们推导出的算法与本文的算法不同。

8. 实验

我们设计实验以研究以下问题：

[1] 此处解释来自 Chatgpt

1. 单路径（single path）和藤蔓（vine）采样程序的性能特征是什么？
2. TRPO 与先前的方法（例如自然策略梯度）相关，但做了几处修改，最显著的是使用固定的 KL 散度约束而非固定的惩罚系数。这对算法的性能有何影响？
3. TRPO 能否用于解决具有挑战性的大规模问题？在最终性能、计算时间和样本复杂度方面，TRPO 在应用于大规模问题时与其他方法相比如何？

为了回答 (1) 和 (2)，我们比较了 TRPO 的单路径（single path）和藤蔓（vine）变体、几种消融变体以及若干先前的策略优化算法的性能。关于 (3)，我们展示了单路径和藤蔓算法能够从零开始获得高质量的运动控制器，这被认为是一个难题。我们还展示了这些算法在使用具有数万个参数的卷积神经网络从图像中学习玩 Atari 游戏的策略时，产生了有竞争力的结果。

8.1 模拟机器人运动

我们使用 MuJoCo 模拟器[15] 进行了机器人运动实验。三个模拟机器人如图 2 所示。机器人的状态是它们的广义位置和速度，控制输入是关节力矩。欠驱动、高维度以及由于接触导致的非平滑动力学使得这些任务极具挑战性。我们的评估包含以下模型：

1. **Swimmer（游泳者）**。10 维状态空间，奖励函数为前进速度的线性奖励加上关节努力的二次惩罚，即 $r(x,u) = v_x - 10^{-5}\|u\|^2$。游泳者可以通过做波浪运动推动自己前进。
2. **Hopper（单足跳跃者）**。12 维状态空间，奖励函数与游泳者相同，但在非终止状态额外奖励 +1。当单足跳跃者摔倒时（由躯干高度和角度的阈值定义），我们终止该回合。
3. **Walker（双足步行者）**。18 维状态空间。对于双足步行者，我们添加了对脚部与地面强烈碰撞的惩罚，以鼓励平滑行走而非跳跃步态。

我们在所有实验中使用 $\delta = 0.01$。关于实验设置和使用参数的更多细节，请参见附录中的表 2。我们使用神经网络表示策略，其架构如图 3 所示，更多细节在附录 D 中提供。为了建立标准基线，我们还包含了经典的倒立摆平衡问题，基于[16] 的表述，使用具有六个参数的线性策略，该策略易于通过无导数黑盒优化方法进行优化。

比较中考虑了以下算法：单路径 TRPO（single path TRPO）；藤蔓 TRPO（vine TRPO）；交叉熵方法（cross-entropy method, CEM），一种无梯度方法[17]；协方差矩阵自适应（covariance matrix adaptation, CMA），另一种无梯度方法[18]；自然梯度（natural gradient），经典的自然策略梯度算法[2]，它与单路径 TRPO 的区别在于使用

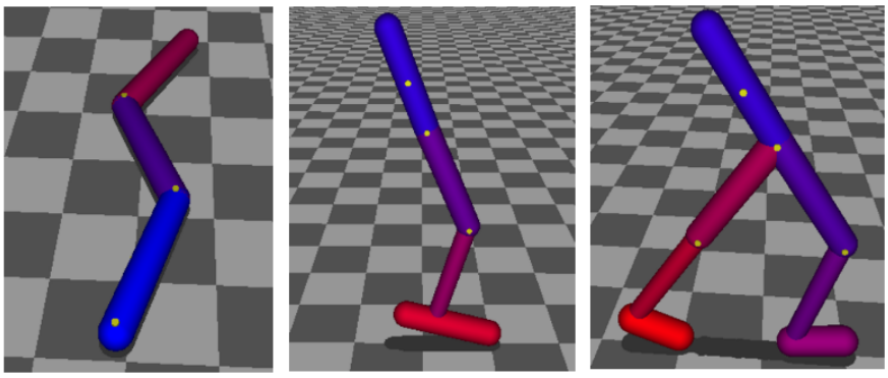


图 0.50　用于运动实验的二维机器人模型。从左到右：游泳者、跳跃者、行走者。跳跃者和行走者因欠驱动和接触不连续性存在特殊挑战。

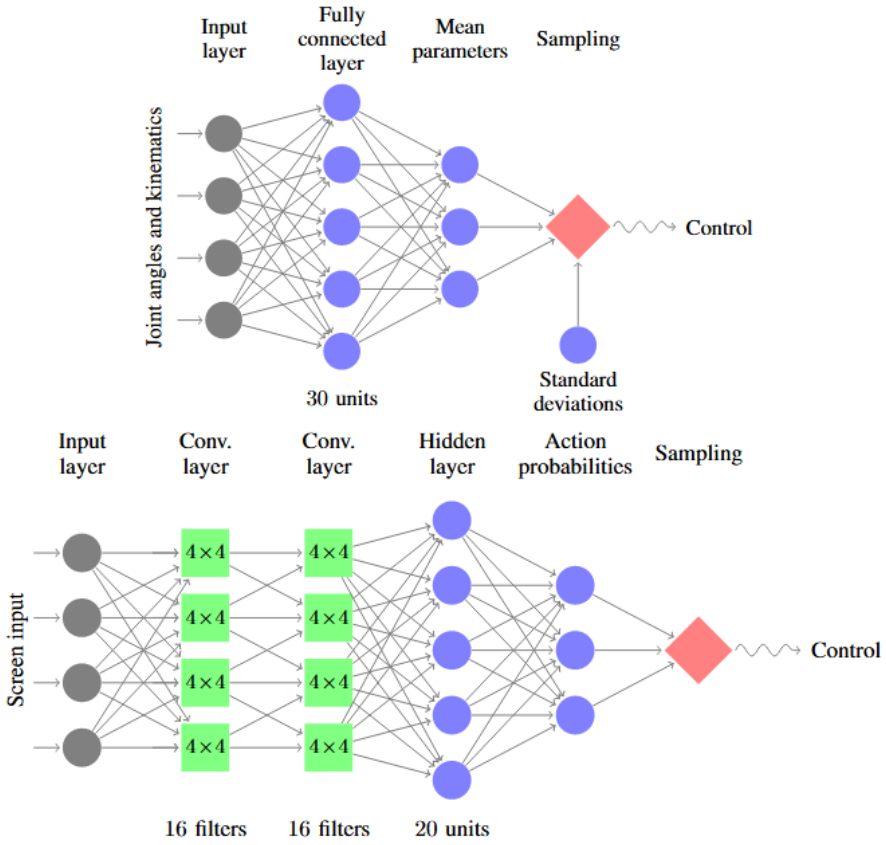


图 0.51　用于运动任务（上图）和玩 Atari 游戏（下图）的神经网络。

固定的惩罚系数（拉格朗日乘子）而非 KL 散度约束；经验 FIM（empirical FIM），与单路径 TRPO 相同，但 FIM 是使用梯度的协方差矩阵而非解析估计来估计的；最大 KL（max KL），该方法仅在倒立摆问题上可行，使用方程 (11) 中的最大 KL 散度而非平均散度，使我们能够评估这种近似的质量。实验中使用的参数在附录 E 中提供。对于自然梯度方法，我们以三倍的步长扫过可能的值，并根据最终性能选取最佳值。图 4 显示了每种算法五次运行的平均总奖励的学习曲线。单路径和

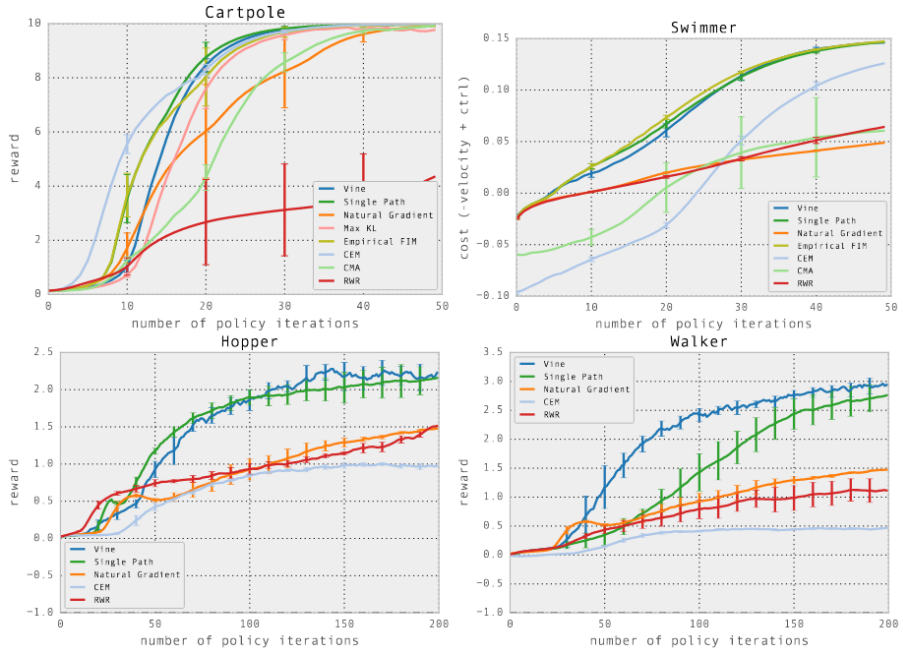


图 0.52　运动任务的学习曲线，每种算法在五次随机初始化运行中的平均值。注意：对于跳跃者和行走者，即使没有任何前进速度也能达到-1 的分数，这表明策略仅仅学会了平衡站立，而没有学会行走。

藤蔓 TRPO 解决了所有问题，产生了最佳解决方案。自然梯度方法在两个较简单的问题上表现良好，但无法产生向前移动的跳跃和行走步态。这些结果提供了经验证据，表明与使用固定的惩罚系数相比，约束 KL 散度是一种更鲁棒的选择步长和实现快速、一致进展的方法。CEM 和 CMA 是无导数算法，因此其样本复杂度随参数数量的增加而呈不利变化，并且在较大规模问题上表现不佳。最大 KL 方法由于约束形式更为严格，学习速度比我们的最终方法稍慢，但总体结果表明，平均 KL 散度约束与理论依据充分的最大 KL 散度具有相似的效果。由 TRPO 学习到的策略视频可在项目网站上查看：`http://sites.google.com/site/trpopaper/`。

需要注意的是，TRPO 使用通用策略和简单的奖励函数学习了所有步态，仅使用了最少的先验知识。这与大多数先前的运动学习方法形成对比，后者通常依赖于手工设计的策略类，这些策略类明确编码了平衡和迈步的概念[19–21]。

8.2 从图像中玩游戏

为了在具有复杂观测的部分可观测任务上评估 TRPO，我们训练了玩 Atari 游戏的策略，使用原始图像作为输入。这些游戏需要学习各种行为，例如躲避子弹和使用球拍击球。除了高维度外，这些游戏的挑战性要素还包括：延迟奖励（在《Breakout》或《Space Invaders》中失去生命时不会立即受到惩罚）；复杂的行为序列（《Q*bert》要求角色在 21 个不同的平台上跳跃）；以及非平稳的图像统计特性（《Enduro》涉及变化和闪烁的背景）。

我们在[1]和[22]报告的相同七款游戏上测试了我们的算法，这些游戏通过 Arcade Learning Environment[23] 提供。图像预处理遵循[1]的协议，策略由图 3 所示的卷积神经网络表示，该网络包含两个具有 16 个通道和步长为 2 的卷积层，后接一个具有 20 个单元的全连接层，共产生 33,500 个参数。

藤蔓（vine）和单路径（single path）算法的结果总结在表 1 中，该表还包括了人类专家表现和两种近期方法：深度 Q 学习[1]，以及蒙特卡洛树搜索与监督训练相结合的方法[22]（称为 UCC-I）。我们的算法在 16 核计算机上运行 500 次迭代大约需要 30 小时（不同游戏间略有差异）。虽然我们的方法仅在部分游戏中优于先前的方法，但它始终取得了合理的分数。与先前的方法不同，我们的方法并非专门为此任务设计。能够将相同的策略搜索方法应用于机器人运动控制和基于图像的游戏玩法等多样化任务，证明了 TRPO 的通用性。

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	-20.4	157	110	179
Human (Mnih et al., 2013)	7456	31.0	368	-3.0	18900	28010	3690
Deep Q Learning (Mnih et al., 2013)	4092	168.0	470	20.0	1952	1705	581
UCC-I (Guo et al., 2014)	5702	380	741	21	20025	2995	692
TRPO - single path	1425.2	10.8	534.6	20.9	1973.5	1908.6	568.4
TRPO - vine	859.5	34.2	430.8	20.9	7732.5	788.4	450.2

表 21　性能比较：基于视觉的 RL 算法在 Atari 任务上的表现。我们的算法（底部两行）在每个任务上仅运行一次，使用相同的架构和参数。由于策略的随机初始化不同，性能在不同运行之间差异较大，但由于时间限制，我们未能获得误差统计。

9. 讨论

我们提出并分析了用于优化随机控制策略的信赖域方法。我们证明了一种算法的单调改进性，该算法使用 KL 散度惩罚反复优化策略期望回报的局部近似，并且我们展示了该方法的一种结合 KL 散度约束的近似在一系列具有挑战性的策略学习任务上取得了良好的实证结果，优于先前的方法。我们的分析还提供了一个统一策略梯度和策略迭代方法的视角，并表明它们是受信赖域约束的某种目标优化算法的特殊极限情况。

在机器人运动领域，我们使用通用神经网络和最小信息奖励，在物理模拟器

中成功学习了游泳、行走和跳跃的控制器。据我们所知，尚未有先前的工作使用通用的策略搜索方法和非工程化的通用策略表示，从头开始学习所有这些任务的控制器。

在游戏玩法领域，我们学习了使用原始图像作为输入的卷积神经网络策略。这需要优化极高维度的策略，而此前只有两种方法报告了在此任务上的成功结果。

由于我们提出的方法具有可扩展性和坚实的理论基础，我们希望它能成为未来工作的一个起点，用于训练大规模、丰富的函数逼近器以解决一系列具有挑战性的问题。在我们探索的两个实验领域的交叉点上，存在学习使用视觉和原始感官数据作为输入的机器人控制策略的可能性，这为训练同时执行感知和控制任务的机器人控制器提供了一个统一的方案。使用更复杂的策略，包括具有隐藏状态的循环策略，可以进一步在部分可观测设置中将状态估计和控制整合到同一策略中。通过将我们的方法与模型学习相结合，还可以显著降低其样本复杂性，使其适用于样本获取成本高昂的现实世界设置。

参考文献

[1] 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.

[2] Sham Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems*, pages 1057–1063. MIT Press, 2002.

[3] David R Hunter and Kenneth Lange. A tutorial on mm algorithms. *The American Statistician*, 58(1):30–37, 2004.

[4] J. A. Bagnell and J. Schneider. Covariant policy search. In *IJCAI*, 2003.

[5] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

[6] J. Peters, K. Mülling, and Y. Altün. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence*, 2010.

[7] P. L. Bartlett and J. Baxter. Infinite-horizon policy-gradient estimation. *arXiv preprint arXiv:1106.0665*, 2011.

[8] Michail G Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *ICML*, volume 3, pages 424–431, 2003.

[9] Victor Gabillon, Mohammad Ghavamzadeh, and Bruno Scherrer. Approximate dynamic programming finally performs well in the game of tetris. In *Advances in Neural Information Processing Systems*, 2013.

[10] D. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific, 2005.

[11] A. Y. Ng and M. Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Uncertainty in artificial intelligence (UAI)*, 2000.

[12] Art B. Owen. Monte carlo theory, methods and examples, 2013.

[13] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[14] Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. Safe policy iteration. In *Proceedings of The 30th International Conference on Machine Learning*, pages 307–315, 2013.

[15] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[16] A. Barto, R. Sutton, and C. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, (5):834–846, 1983.

[17] István Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941, 2006.

[18] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317. IEEE, 1996.

[19] R. Tedrake, T. Zhang, and H. Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[20] T. Geng, B. Porr, and F. Wörgötter. Fast biped walking with a reflexive controller and realtime policy searching. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[21] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Transactions on Graphics (TOG)*, 28:60, 2009.

[22] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline montecarlo tree search planning. In *Advances in Neural Information Processing Systems*, pages 3338–3346, 2014.

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[24] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2009.

[25] J. Martens and I. Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural Networks: Tricks of the Trade*, pages 479–535. Springer, 2012.

[26] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[27] Stephen J Wright and Jorge Nocedal. *Numerical optimization*, volume 2. Springer New York, 1999.

A. 策略改进界限的证明

本证明（定理 1 的证明）借鉴了[2] 中定理 4.1 证明的技术，并将其应用于本文考虑的更一般设定。非正式概述如下。我们的证明依赖于耦合（coupling）的概念，即我们联合定义策略 π 和 π'，使得它们以高概率 $= (1-\alpha)$ 选择相同的动作。代理

损失 $L_\pi(\tilde{\pi})$ 考虑了 $\tilde{\pi}$ 第一次与 π 不一致时的优势，但不考虑后续的不一致。因此，L_π 中的误差是由于 π 和 $\tilde{\pi}$ 之间发生两次或更多次不一致造成的，因此我们得到了一个 $O(\alpha^2)$ 的修正项，其中 α 是不一致的概率。

我们首先介绍来自[2] 的一个引理，该引理表明策略性能差异 $\eta(\tilde{\pi}) - \eta(\pi)$ 可以分解为每个时间步优势的求和。

给定两个策略 $\pi, \tilde{\pi}$，

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t, a_t)\right] \tag{19}$$

该期望是对轨迹 $\tau := (s_0, a_0, s_1, a_0, \ldots)$ 取的，符号 $\mathbb{E}_{\tau\sim\tilde{\pi}}[\ldots]$ 表示动作是从 $\tilde{\pi}$ 中采样以生成 τ 的。

证明： 首先注意 $A_\pi(s,a) = \mathbb{E}_{s'\sim P(s'|s,a)}[r(s) + \gamma V_\pi(s') - V_\pi(s)]$。因此，

$$\mathbb{E}_{\tau|\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t, a_t)\right] \tag{20}$$

$$= \mathbb{E}_{\tau|\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t (r(s_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t))\right] \tag{21}$$

$$= \mathbb{E}_{\tau|\tilde{\pi}}\left[-V_\pi(s_0) + \sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \tag{22}$$

$$= -\mathbb{E}_{s_0}[V_\pi(s_0)] + \mathbb{E}_{\tau|\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \tag{23}$$

$$= -\eta(\pi) + \eta(\tilde{\pi}) \tag{24}$$

重新整理，即得结果。 □

定义 $\bar{A}(s)$ 为在状态 s 处 $\tilde{\pi}$ 相对于 π 的期望优势：

$$\bar{A}(s) = \mathbb{E}_{a\sim\tilde{\pi}(\cdot|s)}\left[A_\pi(s,a)\right]. \tag{25}$$

现在，引理 可以写作如下形式：

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t \bar{A}(s_t)\right]. \tag{26}$$

注意 L_π 可以写作：

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t \bar{A}(s_t)\right]. \tag{27}$$

这些等式的区别在于状态是使用 π 还是 $\tilde{\pi}$ 采样的。为了界定 $\eta(\tilde{\pi})$ 和 $L_\pi(\tilde{\pi})$ 之间的差异，我们将界定每个时间步产生的差异。为此，我们首先需要引入一个度

量 π 和 $\tilde{\pi}$ 之间一致程度的指标。具体来说，我们将耦合（couple）这两个策略，使得它们定义了一个关于动作对的联合分布。

定义 2. $(\pi, \tilde{\pi})$ 是一个 α-耦合策略对，如果它定义了一个联合分布 $(a, \tilde{a})|s$，使得对于所有 s，有 $P(a \neq \tilde{a}|s) \leq \alpha$。$\pi$ 和 $\tilde{\pi}$ 将分别表示 a 和 $\tilde{a}$ 的边缘分布。

从计算角度讲，α-耦合意味着如果我们随机选择一个随机数生成器的种子，然后在设置该种子后分别从 π 和 $\tilde{\pi}$ 中采样，那么对于至少 $1-\alpha$ 比例的种子，采样结果将是一致的。

给定 π 和 $\tilde{\pi}$ 是 α-耦合策略，则对于所有状态 s，

$$\left|\bar{A}(s)\right| \leq 2\alpha \max_{s,a} |A_\pi(s,a)|. \tag{28}$$

证明：

$$\bar{A}(s) = \mathbb{E}_{\tilde{a}\sim\tilde{\pi}}\left[A_\pi(s,\tilde{a})\right] = \mathbb{E}_{(a,\tilde{a})\sim(\pi,\tilde{\pi})}\left[A_\pi(s,\tilde{a}) - A_\pi(s,a)\right] \quad \text{因为} \quad \mathbb{E}_{a\sim\pi}\left[A_\pi(s,a)\right] = 0 \tag{29}$$

$$= P(a \neq \tilde{a}|s)\mathbb{E}_{(a,\tilde{a})\sim(\pi,\tilde{\pi})|a\neq\tilde{a}}\left[A_\pi(s,\tilde{a}) - A_\pi(s,a)\right] \tag{30}$$

$$|\bar{A}(s)| \leq \alpha \cdot 2 \max_{s,a} |A_\pi(s,a)| \tag{31}$$

□

令 $(\pi, \tilde{\pi})$ 是一个 α-耦合策略对。则

$$\left|\mathbb{E}_{s_t\sim\tilde{\pi}}\left[\bar{A}(s_t)\right] - \mathbb{E}_{s_t\sim\pi}\left[\bar{A}(s_t)\right]\right| \leq 2\alpha \max_s \bar{A}(s) \leq 4\alpha(1-(1-\alpha)^t) \max_{s,a} |A_\pi(s,a)|. \tag{32}$$

证明： 给定耦合策略对 $(\pi, \tilde{\pi})$，我们也可以获得由 π 和 $\tilde{\pi}$ 分别产生的轨迹分布的耦合。即，我们有轨迹对 $(\tau, \tilde{\tau})$，其中 τ 是通过从 π 采取动作获得的，$\tilde{\tau}$ 是通过从 $\tilde{\pi}$ 采取动作获得的，其中使用相同的随机种子生成两个轨迹。我们将考虑在时间步 t 处 $\tilde{\pi}$ 相对于 π 的优势，并根据 π 和 $\tilde{\pi}$ 在所有时间步 $i < t$ 是否一致来分解此期望。

令 n_t 表示在 $i < t$ 时 $a_i \neq \tilde{a}_i$ 的次数，即在时间步 t 之前 π 和 $\tilde{\pi}$ 不一致的次数。

$$\mathbb{E}_{s_t\sim\tilde{\pi}}\left[\bar{A}(s_t)\right] = P(n_t=0)\mathbb{E}_{s_t\sim\tilde{\pi}|n_t=0}\left[\bar{A}(s_t)\right] + P(n_t>0)\mathbb{E}_{s_t\sim\tilde{\pi}|n_t>0}\left[\bar{A}(s_t)\right] \tag{33}$$

$$\mathbb{E}_{s_t\sim\pi}\left[\bar{A}(s_t)\right] = P(n_t=0)\mathbb{E}_{s_t\sim\pi|n_t=0}\left[\bar{A}(s_t)\right] + P(n_t>0)\mathbb{E}_{s_t\sim\pi|n_t>0}\left[\bar{A}(s_t)\right] \tag{34}$$

注意 $n_t = 0$ 的项是相等的：

$$\mathbb{E}_{s_t\sim\tilde{\pi}|n_t=0}\left[\bar{A}(s_t)\right] = \mathbb{E}_{s_t\sim\pi|n_t=0}\left[\bar{A}(s_t)\right], \tag{35}$$

因为 $n_t = 0$ 表示 π 和 $\tilde{\pi}$ 在所有小于 t 的时间步上一致。将方程 (33) 和 (34) 相减，

我们得到

$$\mathbb{E}_{s_t\sim\tilde{\pi}}\left[\bar{A}(s_t)\right]-\mathbb{E}_{s_t\sim\pi}\left[\bar{A}(s_t)\right]=P(n_t>0)\left(\mathbb{E}_{s_t\sim\tilde{\pi}|n_t>0}\left[\bar{A}(s_t)\right]-\mathbb{E}_{s_t\sim\pi|n_t>0}\left[\bar{A}(s_t)\right]\right) \tag{36}$$

根据 α 的定义，$P(\pi,\tilde{\pi}$ 在时间步 i 一致$)\geq 1-\alpha$，所以 $P(n_t=0)\geq(1-\alpha)^t$，且

$$P(n_t>0)\leq 1-(1-\alpha)^t \tag{37}$$

接下来，注意

$$\left|\mathbb{E}_{s_t\sim\tilde{\pi}|n_t>0}\left[\bar{A}(s_t)\right]-\mathbb{E}_{s_t\sim\pi|n_t>0}\left[\bar{A}(s_t)\right]\right|\leq\left|\mathbb{E}_{s_t\sim\tilde{\pi}|n_t>0}\left[\bar{A}(s_t)\right]\right|+\left|\mathbb{E}_{s_t\sim\pi|n_t>0}\left[\bar{A}(s_t)\right]\right| \tag{38}$$

$$\leq 4\alpha\max_{s,a}|A_\pi(s,a)| \tag{39}$$

其中第二个不等式由引理 得出。

将方程 (37) 和方程 (39) 代入方程 (36)，我们得到

$$\left|\mathbb{E}_{s_t\sim\tilde{\pi}}\left[\bar{A}(s_t)\right]-\mathbb{E}_{s_t\sim\pi}\left[\bar{A}(s_t)\right]\right|\leq 4\alpha(1-(1-\alpha)^t)\max_{s,a}|A_\pi(s,a)| \tag{40}$$

□

前面的引理界定了每个时间步 t 处期望优势的差异。我们可以通过对时间求和来界定 $\eta(\tilde{\pi})$ 和 $L_\pi(\tilde{\pi})$ 之间的差异。将方程 (26) 与方程 (27) 相减，并定义 $\epsilon=\max_{s,a}|A_\pi(s,a)|$，可得：

$$|\eta(\tilde{\pi})-L_\pi(\tilde{\pi})|=\sum_{t=0}^{\infty}\gamma^t\left|\mathbb{E}_{\tau\sim\tilde{\pi}}\left[\bar{A}(s_t)\right]-\mathbb{E}_{\tau\sim\pi}\left[\bar{A}(s_t)\right]\right| \tag{41}$$

$$\leq\sum_{t=0}^{\infty}\gamma^t\cdot 4\epsilon\alpha(1-(1-\alpha)^t) \tag{42}$$

$$=4\epsilon\alpha\left(\frac{1}{1-\gamma}-\frac{1}{1-\gamma(1-\alpha)}\right) \tag{43}$$

$$=\frac{4\alpha^2\gamma\epsilon}{(1-\gamma)(1-\gamma(1-\alpha))} \tag{44}$$

$$\leq\frac{4\alpha^2\gamma\epsilon}{(1-\gamma)^2} \tag{45}$$

最后，为了将 α 替换为总变分散度（total variation divergence），我们需要利用总变分散度与耦合随机变量之间的对应关系：

> 假设 p_X 和 p_Y 是满足 $D_{\mathrm{TV}}(p_X\parallel p_Y)=\alpha$ 的分布。那么存在一个联合分布 (X,Y)，其边缘分布为 p_X,p_Y，且 $X=Y$ 的概率为 $1-\alpha$。

参见[24] 的命题 4.7。由此可知，如果我们有两个策略 π 和 $\tilde{\pi}$，满足 $\max_s D_{\mathrm{TV}}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s)) \leq \alpha$，那么我们可以定义一个具有相应边缘分布的 α-耦合策略对 $(\pi, \tilde{\pi})$。将 $\alpha = \max_s D_{\mathrm{TV}}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s))$ 代入方程 (45)，定理 (1) 即得证。

附录 B 策略改进界限的摄动理论证明

我们使用摄动理论提供了定理1的另一种证明。

证明： 令 $G = (1 + \gamma P_\pi + (\gamma P_\pi)^2 + \ldots) = (1 - \gamma P_\pi)^{-1}$，类似地令 $\tilde{G} = (1 + \gamma P_{\tilde{\pi}} + (\gamma P_{\tilde{\pi}})^2 + \ldots) = (1 - \gamma P_{\tilde{\pi}})^{-1}$。我们将采用以下约定：$\rho$（状态空间上的密度）是一个向量，而 r（状态空间上的奖励函数）是一个对偶向量（即向量上的线性泛函），因此 $r\rho$ 是一个标量，表示在密度 ρ 下的期望奖励。注意 $\eta(\pi) = rG\rho_0$，且 $\eta(\tilde{\pi}) = r\tilde{G}\rho_0$。令 $\Delta = P_{\tilde{\pi}} - P_\pi$。我们希望界定 $\eta(\tilde{\pi}) - \eta(\pi) = r(\tilde{G} - G)\rho_0$。

$$
\begin{aligned}
\eta(\pi) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{\tau \sim \pi} \left[r(s_t) \right] \\
&= \sum_{t=0}^{\infty} \gamma^t r \rho_t \\
&= \sum_{t=0}^{\infty} \gamma^t r (P_\pi)^t \rho_0 \\
&= rG\rho_0
\end{aligned}
$$

我们从一些标准的摄动理论操作开始。

$$
\begin{aligned}
G^{-1} - \tilde{G}^{-1} &= (1 - \gamma P_\pi) - (1 - \gamma P_{\tilde{\pi}}) \\
&= \gamma \Delta
\end{aligned} \tag{46}
$$

左乘 G 并右乘 $\tilde{G}$：

$$
\begin{aligned}
\tilde{G} - G &= \gamma G \Delta \tilde{G} \\
\tilde{G} &= G + \gamma G \Delta \tilde{G}
\end{aligned} \tag{47}
$$

将右侧代入 $\tilde{G}$ 得到：

$$
\tilde{G} = G + \gamma G \Delta G + \gamma^2 G \Delta G \Delta \tilde{G} \tag{48}
$$

因此我们有：

$$
\eta(\tilde{\pi}) - \eta(\pi) = r(\tilde{G} - G)\rho = \gamma r G \Delta G \rho_0 + \gamma^2 r G \Delta G \Delta \tilde{G} \rho_0 \tag{49}
$$

让我们首先考虑第一项 $\gamma r G \Delta G \rho_0$。注意 $rG = v$，即无限视界状态价值函数。同时注意 $G\rho_0 = \rho_\pi$。因此我们可以写出 $\gamma r G \Delta G \rho_0 = \gamma v \Delta \rho_\pi$。我们将证明这个表达式

等于期望优势 $L_\pi(\tilde{\pi}) - L_\pi(\pi)$.

$$
\begin{aligned}
L_\pi(\tilde{\pi}) - L_\pi(\pi) &= \sum_s \rho_\pi(s) \sum_a (\tilde{\pi}(a \mid s) - \pi(a \mid s)) A_\pi(s,a) \\
&= \sum_s \rho_\pi(s) \sum_a \left(\tilde{\pi}(a \mid s) - \pi(a \mid s)\right) \left[r(s) + \sum_{s'} p(s' \mid s,a) \gamma v(s') - v(s) \right] \\
&= \sum_s \rho_\pi(s) \sum_a \left(\tilde{\pi} - \pi\right) r(s) + \\
&\sum_s \rho_\pi(s) \sum_a \left(\tilde{\pi} - \pi\right) \sum_{s'} p(s' \mid s,a) \gamma v(s') - \sum_s \rho_\pi(s) \sum_a \left(\tilde{\pi} - \pi\right) v(s) \\
&= \sum_s \rho_\pi(s) \sum_{s'} \sum_a \left(\tilde{\pi}(a \mid s) - \pi(a \mid s)\right) p(s' \mid s,a) \gamma v(s') \\
&= \sum_s \rho_\pi(s) \sum_{s'} \left(p_\pi(s' \mid s) - p_{\tilde{\pi}}(s' \mid s)\right) \gamma v(s') \\
&= \gamma v \Delta \rho_\pi
\end{aligned} \tag{50}
$$

接下来我们界定 $O(\Delta^2)$ 项 $\gamma^2 r G \Delta G \Delta \tilde{G} \rho$. 首先我们考虑乘积 $\gamma r G \Delta = \gamma v \Delta$。考虑这个对偶向量的第 s 个分量

$$
\begin{aligned}
|(\gamma v \Delta)_s| &= \left| \gamma \sum_{s'} V(s') \sum_a (\tilde{\pi}(a \mid s) - \pi(a \mid s)) P(s'|s,a) \right| \\
&= \left| \sum_a (\tilde{\pi}(a \mid s) - \pi(a \mid s)) [r(s) + \gamma \sum_{s'} P(s'|s,a) V(s')] \right| \\
&= \left| \sum_a (\tilde{\pi}(a \mid s) - \pi(a \mid s)) Q_\pi(s,a) \right| \\
&= \left| \sum_a (\tilde{\pi}(a \mid s) - \pi(a \mid s)) A_\pi(s,a) \right| \\
&\leq \sum_a |\tilde{\pi}(a \mid s) - \pi(a \mid s)| \cdot \max_a |A_\pi(s,a)| \\
&\leq 2\alpha\epsilon
\end{aligned} \tag{51}
$$

其中最后一行使用了总变分散度的定义，以及 $\epsilon = \max_{s,a} |A_\pi(s,a)|$ 的定义。我们使用 ℓ_1 算子范数来界定另一部分 $G\Delta\tilde{G}\rho$：

$$
\|A\|_1 = \sup_\rho \left\{ \frac{\|A\rho\|_1}{\|\rho\|_1} \right\} \tag{52}
$$

其中我们有 $\|G\|_1 = \|\tilde{G}\|_1 = 1/(1-\gamma)$ 且 $\|\Delta\|_1 = 2\alpha$。由此可得：

$$
\|G\|_1 = \| \sum_{t=0}^{\infty} \gamma^t P_\pi^t \|_1 \leq \| \sum_{t=0}^{\infty} \gamma^t \|_1 \| P_\pi^t \|_1 = \frac{1}{1-\gamma}
$$

$$
\begin{aligned}
\|\Delta\|_\infty &= \max_j \sum_i |P(S'=s_i|S=s_j,\tilde{\pi}) - P(S'=s_i|S=s_j,\pi)| \\
&= \max_j \sum_i |\sum_a P(S'=s_i|S=s_j,a)[\tilde{\pi}(a|s_j) - \pi(a|s_j)]| \\
&\le \max_j \sum_a |\tilde{\pi}(a|s_j) - \pi(a|s_j)| \\
&= 2\alpha
\end{aligned}
$$

$$
\begin{aligned}
\|G\Delta\tilde{G}\rho\|_1 &\le \|G\|_1\|\Delta\|_1\|\tilde{G}\|_1\|\rho\|_1 \\
&= \frac{1}{1-\gamma}\cdot 2\alpha \cdot \frac{1}{1-\gamma}\cdot 1
\end{aligned} \tag{53}
$$

因此我们有：

$$
\begin{aligned}
\gamma^2\left|rG\Delta G\Delta\tilde{G}\rho\right| &\le \gamma\|\gamma rG\Delta\|_\infty\|G\Delta\tilde{G}\rho\|_1 \\
&\le \gamma\|v\Delta\|_\infty\|G\Delta\tilde{G}\rho\|_1 \\
&\le \gamma\cdot 2\alpha\epsilon\cdot\frac{2\alpha}{(1-\gamma)^2} \\
&= \frac{4\gamma\epsilon}{(1-\gamma)^2}\alpha^2
\end{aligned} \tag{54}
$$

□

C. 高效求解信赖域约束优化问题

本节描述如何高效地近似求解以下约束优化问题，该问题需要在 TRPO 的每次迭代中求解：

$$
\text{maximize } L(\theta) \quad \text{subject to } \overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}},\theta) \le \delta. \tag{55}
$$

我们将描述的方法涉及两个步骤：(1) 计算搜索方向，使用目标函数的线性近似和约束的二次近似；(2) 在该方向上进行线搜索，确保我们改进非线性目标同时满足非线性约束。

搜索方向通过近似求解方程 $Ax = g$ 来计算，其中 A 是 Fisher 信息矩阵，即 KL 散度约束的二次近似：$\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}},\theta) \approx \frac{1}{2}(\theta-\theta_{\mathrm{old}})^T A(\theta-\theta_{\mathrm{old}})$，其中 $A_{ij} = \frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}},\theta)$。在大规模问题中，形成完整的矩阵 A（或 A^{-1}）在计算和内存方面成本高昂。然而，共轭梯度算法[1] 允许我们在仅能计算矩阵-向量乘积 $y \to Ay$ 的函数时，无需形成完整矩阵即可近似求解方程 $Ax = b$。附录 C.1 描述了计算 Fisher 信息矩阵的矩阵-向量乘积的最有效方法。关于使用 Hessian-向量乘积优化神经网

[1]推荐阅读 Wikipedia Conjugate gradient method，链接：`https://en.wikipedia.org/wiki/Conjugate_gradient_method`

络目标的更多阐述，请参见[25] 和[26]。

需要计算最大步长 β，使得 $\theta + \beta s$ 满足 KL 散度约束。为此，令 $\delta = \overline{D}_{\mathrm{KL}} \approx \frac{1}{2}(\beta s)^T A(\beta s) = \frac{1}{2}\beta^2 s^T A s$。由此可得 $\beta = \sqrt{2\delta / s^T A s}$，其中 δ 是期望的 KL 散度。项 $s^T A s$ 可以通过单次 Hessian 向量乘积计算，并且也是共轭梯度算法产生的中间结果。

最后，我们使用线搜索来确保代理目标的改进和 KL 散度约束的满足，这两者在参数向量 θ 上都是非线性的（因此偏离了用于计算步长的线性和二次近似）。我们在线目标 $L_{\theta_{\mathrm{old}}}(\theta) - \mathcal{X}[\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}}, \theta) \leq \delta]$ 上执行线搜索，其中 $\mathcal{X}[\ldots]$ 当其参数为真时等于零，为假时等于 $+\infty$。从上一段计算的最大步长 β 开始，我们指数级收缩 β 直到目标改进。没有这个线搜索，算法偶尔会计算导致性能灾难性下降的大步长。

C.1 计算 Fisher-向量乘积

这里我们将描述如何计算平均 Fisher 信息矩阵与任意向量之间的矩阵-向量乘积。这种矩阵-向量乘积使我们能够执行共轭梯度算法。假设参数化策略从输入 x 映射到“分布参数”向量 $\mu_\theta(x)$，该向量参数化分布 $\pi(u|x)$。现在，对于给定输入 x 的 KL 散度可以写成如下形式：

$$D_{\mathrm{KL}}(\pi_{\theta_{\mathrm{old}}}(\cdot|x) \parallel \pi_\theta(\cdot|x)) = \mathrm{kl}(\mu_\theta(x), \mu_{\mathrm{old}}) \tag{56}$$

其中 kl 是两个均值参数向量对应分布之间的 KL 散度。对 θ 求二阶导数，我们得到：

$$\frac{\partial \mu_a(x)}{\partial \theta_i} \frac{\partial \mu_b(x)}{\partial \theta_j} \mathrm{kl}''_{ab}(\mu_\theta(x), \mu_{\mathrm{old}}) + \frac{\partial^2 \mu_a(x)}{\partial \theta_i \partial \theta_j} \mathrm{kl}'_a(\mu_\theta(x), \mu_{\mathrm{old}}) \tag{57}$$

其中撇号 ($\prime$) 表示对第一个参数的微分，并且对索引 a, b 有隐含求和。第二项消失，只留下第一项。令 $J := \frac{\partial \mu_a(x)}{\partial \theta_i}$（雅可比矩阵），则 Fisher 信息矩阵可以写成矩阵形式为 $J^T M J$，其中 $M = \mathrm{kl}''_{ab}(\mu_\theta(x), \mu_{\mathrm{old}})$ 是关于均值参数 μ 的分布的 Fisher 信息矩阵（与参数 θ 相对）。对于大多数感兴趣的参数化分布，这有一个简单的形式。

现在，Fisher-向量乘积可以写成一个函数 $y \to J^T M J y$。乘以 J^T 和 J 可以由大多数自动微分和神经网络包执行（乘以 J^T 是著名的反向传播操作），并且对于感兴趣的分布，可以推导出乘以 M 的操作。注意，这种 Fisher-向量乘积在一组数据点（即 μ 的输入 x）上计算平均是容易的。

或者，可以使用通用方法通过反向模式自动微分计算 Hessian-向量乘积（[27]，第 8 章），计算 $\overline{D}_{\mathrm{KL}}$ 关于 θ 的 Hessian。这种方法效率稍低，因为它没有利用 $\mu(x)$

的二阶导数（即方程 (57) 中的第二项）可以忽略的事实，但可能实现起来大体上更容易。

我们已经描述了计算 Fisher-向量乘积 $y \to Ay$ 的过程，其中 Fisher 信息矩阵在一组输入到函数 μ 的数据上平均。计算 Fisher-向量乘积通常与计算依赖于 $\mu(x)$ 的目标的梯度大约一样昂贵[27]。此外，每次梯度计算中需要进行 k 次 Fisher-向量乘积，其中 k 是我们执行的共轭梯度算法的迭代次数。我们发现 $k = 10$ 非常有效，使用更高的 k 并不会导致策略改进更快。因此，一个朴素的实现会将超过 90% 的计算精力花费在这些 Fisher-向量乘积上。然而，我们可以通过子采样数据来计算 Fisher-向量乘积来大大减轻这个负担。由于 Fisher 信息矩阵仅作为一个度量，它可以在数据子集上计算而不会严重降低最终步骤的质量。因此，我们可以在 10% 的数据上计算它，而 Hessian-向量乘积的总成本将与计算梯度大致相同。通过这种优化，自然梯度步骤 $A^{-1}g$ 的计算不会产生超出计算梯度 g 的显著额外计算成本。

D. 使用神经网络近似分解策略

策略，即条件概率分布 $\pi_\theta(a|s)$，可以通过神经网络进行参数化。该神经网络（确定性地）从状态向量 s 映射到一个向量 μ，该向量指定了动作空间上的分布。然后我们可以计算似然 $p(a|\mu)$ 并采样 $a \sim p(a|\mu)$。

在我们关于连续状态和动作空间的实验中，我们使用了高斯分布，其中协方差矩阵是对角矩阵且与状态无关。一个具有多个全连接（密集）层的神经网络从输入特征映射到高斯分布的均值。一组独立的参数指定了每个元素的对数标准差。更具体地说，参数包括用于计算均值的神经网络的权重和偏置集合 $\{W_i, b_i\}_{i=1}^{L}$，以及一个与 a 维度相同的向量 r（对数标准差）。然后，策略由正态分布 $\mathcal{N}\left(\text{均值} = \text{神经网络}\left(s; \{W_i, b_i\}_{i=1}^{L}\right), \text{标准差} = \exp(r)\right)$ 定义。这里，$\mu = \left[\text{均值}, \text{标准差}\right]$。

对于离散动作（Atari）的实验，我们使用分解的离散动作空间，其中每个因子被参数化为一个分类分布。也就是说，动作由一个整数元组 $(a_1, a_2, \ldots, a_K)$ 组成，其中 $a_k \in \{1, 2, \ldots, N_k\}$，并且每个组件被假设具有一个分类分布，该分布由一个向量 $\mu_k = [p_1, p_2, \ldots, p_{N_k}]$ 指定。因此，μ 被定义为因子参数的连接：$\mu = [\mu_1, \mu_2, \ldots, \mu_K]$，其维度为 $\dim \mu = \sum_{k=1}^{K} N_k$。$\mu$ 的分量通过对输入 s 应用神经网络，然后对每个切片应用 softmax 算子来计算，从而为每个因子产生归一化的概率。

	Swimmer	Hopper	Walker
State space dim.	10	12	20
Control space dim.	2	3	6
Total num. policy params	364	4806	8206
Sim. steps per iter.	50K	1M	1M
Policy iter.	200	200	200
Stepsize ($\bar{D}_{\mathrm{KL}}$)	0.01	0.01	0.01
Hidden layer size	30	50	50
Discount (γ)	0.99	0.99	0.99
Vine: rollout length	50	100	100
Vine: rollouts per state	4	4	4
Vine: Q-values per batch	500	2500	2500
Vine: num. rollouts for sampling	16	16	16
Vine: len. rollouts for sampling	1000	1000	1000
Vine: computation time (minutes)	2	14	40
SP: num. path	50	1000	10000
SP: path len.	1000	1000	1000
SP: computation time	5	35	100

表 22　Experiment Parameters for Continuous Control Tasks

E. 实验参数

	All games
Total num. policy params	33500
Vine: Sim. steps per iter.	400K
SP: Sim. steps per iter.	100K
Policy iter.	500
Stepsize ($\overline{D}_{\mathrm{KL}}$)	0.01
Discount (γ)	0.99
Vine: rollouts per state	≈ 4
Vine: computation time	$\approx$ 30 hrs
SP: computation time	$\approx$ 30 hrs

表 23　Parameters used for Atari domain.

E. Atari 游戏的学习曲线

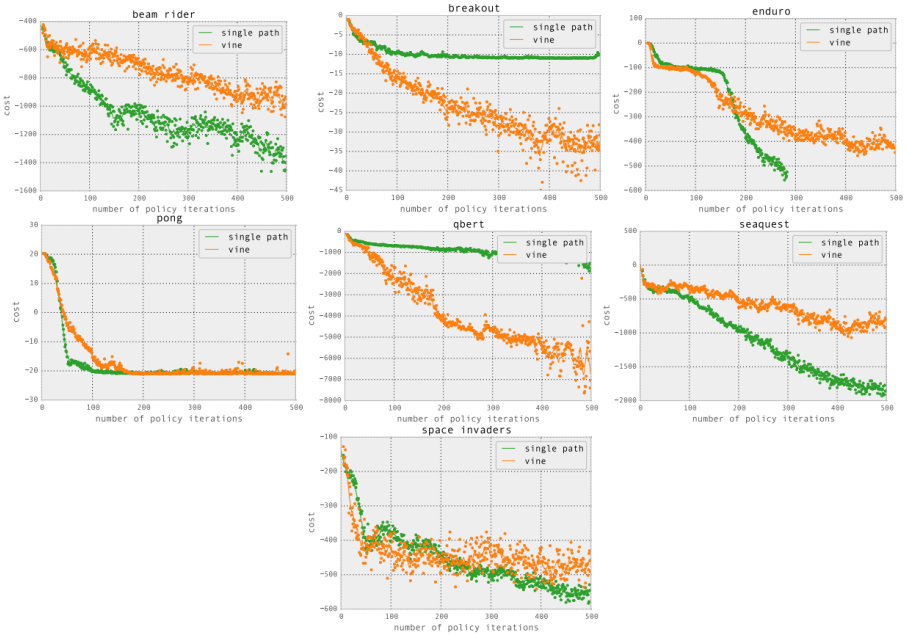


图 0.53　Atari 游戏的学习曲线。由于历史原因，图表显示成本 = 负回报。

高维连续控制中的广义优势估计[1,2]

摘要

在强化学习领域，策略梯度方法是一种颇具吸引力的方案，原因在于它能直接对累积奖励进行优化，且可直接与神经网络等非线性函数近似器结合使用。该方法面临两大主要挑战：一是通常需要大量样本；二是在数据的非平稳性下，如何保持稳定且持续的改进。为解决第一个挑战，我们引入值函数，以引入少量偏差为代价，大幅降低策略梯度估计的方差，我们采用一种类似于（TD (λ)）的优势函数的指数加权估计器。对于第二个挑战，我们采用信任域优化方法，分别对策略和值函数进行优化，这两个函数均由神经网络表示。

我们的方法在极具挑战性的三维运动控制任务上取得了出色的实证结果，包括为双足和四足机器人学习跑步步态，并为双足机器人学习从趴地状态站立起来的策略。与以往大量采用手工设计策略表示的研究不同，我们的神经网络策略能直接将原始运动学数据映射为关节力矩。此外，我们的算法完全是无模型的，且在三维双足机器人学习任务中，所需的仿真经验量对应 1-2 周的真实时间。

1 引言

在强化学习中，典型的问题表述是最大化某一策略的期望总奖励。其中一个关键难点在于：动作的正面或负面影响往往在很久之后才体现在奖励上，这被称为“信用分配问题”（Minsky, 1961；Sutton & Barto, 1998），而在行为科学文献中也被称为“延迟奖励问题”（Hull, 1943）。值函数为这一问题提供了一种简洁的解决方案——它可以在延迟奖励尚未到来之前，估计一个动作的优劣。强化学习算法以多种方式利用值函数。本文聚焦于那些优化参数化策略、并利用值函数辅助估计策略如何改进的算法。

当使用参数化随机策略时，可以获得对期望总回报梯度的无偏估计（Williams, 1992；Sutton，1999；Baxter & Bartlett, 2000），这些带有噪声的梯度估计可用于随机梯度上升算法。遗憾的是，这种梯度估计的方差会随着时间范围的扩大而急剧增加，因为一个动作的效果会与前后动作的影响相互干扰。另一类策略梯度算法

[1]原文：High-Dimensional Continuous Control Using Generalized Advantage Estimation, Schulman et al, 2015. Algorithm: GAE.

[2]译者：丁依宁。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

被称为“演员 - 评论员算法”，使用值函数代替经验回报，从而获得更低方差的估计器，但代价是引入了偏差（Konda & Tsitsiklis, 2003；Hafner & Riedmiller, 2011）。然而，虽然高方差要求更多样本，偏差则更具破坏性——即使样本无限多，偏差也可能导致算法无法收敛，或者收敛到一个远非局部最优的次优解。

为了解决这个问题，我们提出了一类新的策略梯度估计器，在显著降低方差的同时控制偏差在可接受范围内。我们称这一估计方案为广义优势估计（GAE），它由两个参数 $\gamma \in [0,1]$ 和 $\lambda \in [0,1]$ 控制。在在线 Actor-Critic 方法背景下，也有相关方法被提出（Kimura & Kobayashi，1998；Wawrzyński，2009）。但我们提供了更为通用的分析框架，适用于在线与离线设置，并进一步将该方法解释为一种“奖励重塑”的实例（Ng，1999），其中近似值函数被用于对奖励进行重构和引导。

我们在多项极具挑战性的三维运动控制任务上呈现了实验结果。结果显示我们的方法能够在策略和值函数均采用通用神经网络逼近器的情况下，学习出复杂的步态。这些网络参数量超过 10^4，策略直接在力矩级别上控制模拟的三维机器人，其状态维度可达 33，执行器多达 10 个。

本文的研究贡献总结如下：

1. 提出广义优势估计（GAE）并提供理论直觉与合理性分析：尽管该公式曾在先前工作中提出，我们提供了全新的分析，使 GAE 能够应用于更广泛的算法体系中，包括我们实验中使用的离线信任域算法。

2. 提出对值函数使用信任域优化方法：实验发现，该方法在训练拥有成千上万个参数的神经网络值函数时具有鲁棒性与高效性。

3. 结合以上两点，提出一种在高维连续控制任务中表现优异的策略学习算法：该算法能够有效地训练神经网络策略，推动强化学习在高维连续控制领域的最新进展。相关视频可在以下链接查看：https://sites.google.com/site/gaepapersupp。

2 预备知识

我们考虑一种不使用折扣因子的策略优化问题表述。初始状态 s_0 从分布 ρ_0 中采样得到。轨迹 $(s_0, a_0, s_1, a_1, \dots)$，根据策略 $a_t \sim \pi(a_t|s_t)$ 采样动作，并根据动力学模型 $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ 采样状态，直到达到终止（吸收）状态为止。奖励定义为 $r_t = r(s_t, a_t, s_{t+1})$ 每个时间步都会获得奖励。目标是最大化期望总奖励 $\sum_{t=0}^{\infty} r_t$。假设所有策略下这一期望是有限的。需注意，我们在问题表述中没有使用折扣因子，它将在下面作为算法参数出现，用于调整偏差与方差之间的权衡。然而，折扣问题（即最大化 $\sum_{t=0}^{\infty} \gamma^t r_t$）可以通过将折扣因子吸收到奖励函数中，使其变为时间依赖性，从而作为无折扣问题的一种实例来处理。

策略梯度方法通过反复估计梯度 $g := \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$ 来最大化期望总奖励。策略梯度存在多种相关的表达式，其形式如下：

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] \tag{1}$$

其中，Ψ_t 可能是以下情况之一：

1. $\sum_{t=0}^{\infty} r_t$：轨迹的总奖励
2. $\sum_{t'=t}^{\infty} r_{t'}$：执行动作 a_t 之后的奖励
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$：上述公式的基准版本
4. $Q^\pi(s_t, a_t)$：状态-动作价值函数
5. $A^\pi(s_t, a_t)$：优势函数
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$：时序差分残差

后面的公式使用以下定义

$$V^\pi(s_t) := \mathbb{E}\left[\sum_{l=0}^{\infty} r_{t+l} \mid s_t\right] \qquad Q^\pi(s_t, a_t) := \mathbb{E}\left[\sum_{l=0}^{\infty} r_{t+l} \mid s_t, a_t\right] \tag{2}$$

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(优势函数)} \tag{3}$$

此处，$\mathbb{E}$ 的下标列出了需要积分的变量，其中状态与动作分别从动态模型 $P(s_{t+1} \mid s_t, a_t)$ 和策略 $\pi(a_t \mid s_t)$ 中按顺序采样得到。冒号符号 $a : b$ 表示闭区间，对应序列 $(a, a+1, \ldots, b)$。这些公式是众所周知且易于推导的，它们直接由“命题 1”推导得出，该命题将在下文给出。

选择 $\Psi_t = A^\pi(s_t, a_t)$ 能使方差达到近乎最低的水平，但在实际应用中，优势函数是未知的，必须通过估计获得。这一点可以通过以下对策略梯度的直观解释来合理化：沿策略梯度方向更新参数时，应提高“优于平均水平”动作的概率，同时降低“差于平均水平”动作的概率。根据优势函数的定义 $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$，其核心作用是衡量某个动作是否比策略的默认行为好。因此，我们应选择优势函数 $A^\pi(s_t, a_t)$ 作为 Ψ_t，这样一来，梯度项 $\Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)$ 仅在 $A^\pi(s_t, a_t) > 0$ 时，才会指向提高动作概率 $\pi_\theta(a_t|s_t)$ 的方向（即仅强化“优势动作”）。有关策略梯度估计器方差及使用基线的影响的更严格分析，请参见 Greensmith 等人（2004）的研究。

我们将引入一个参数 γ，该参数通过降低延迟效应所对应奖励的权重来减少方差，但代价是会引入偏差。此参数与马尔可夫决策过程（MDP）折扣形式中使用的折扣因子相对应，但在无折扣问题中，我们将其视为方差减少参数；Marbach 与 Tsitsiklis（2003）、Kakade（2001b）以及 Thomas（2014）已从理论层面分析过

该技术。折扣值函数定义如下：

$$V^{\pi,\gamma}(s_t) := \mathbb{E}\left[\sum_{l=0}^{\infty}\gamma^l r_{t+l} \mid s_t\right] \qquad Q^{\pi,\gamma}(s_t,a_t) := \mathbb{E}\left[\sum_{l=0}^{\infty}\gamma^l r_{t+l} \mid s_t,a_t\right] \tag{4}$$

$$A^{\pi,\gamma}(s_t,a_t) := Q^{\pi,\gamma}(s_t,a_t) - V^{\pi,\gamma}(s_t) \tag{5}$$

策略梯度的折扣近似定义如下：

$$g^{\gamma} := \mathbb{E}\left[\sum_{t=0}^{\infty} A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] \tag{6}$$

下一节将讨论如何为折扣优势函数 $A^{\pi,\gamma}$ 构造有偏差（但偏差程度可控）的估计量，进而为公式（6）中的折扣策略梯度提供噪声估计。

在继续往下进行之前，我们将引入 γ- 公正估计量（γ-just）的概念。即在使用该估计器代替未知且必须估计的 $A^{\pi,\gamma}$ 时，不会引入偏差的估计器，用于在方程（6）中估计 g^{γ}。考虑一个优势估计器 $\hat{A}_t(s_{0:\infty},a_{0:\infty})$，它通常是整个轨迹的函数。

定义 1：若估计量 $\hat{A}_t$ 满足以下等式，则称其为 γ- 公正估计量：

$$\mathbb{E}\left[\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] = \mathbb{E}\left[A^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] \tag{7}$$

由此可直接得出：若对所有时刻 t，估计量 $\hat{A}_t$ 均为 γ- 公正估计量，则有：

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] = g^{\gamma} \tag{8}$$

估计量 $\hat{A}_t$ 为 γ- 公正估计量的一个充分条件是：$\hat{A}_t$ 可分解为两个函数 Q_t 与 b_t 的差值。其中，Q_t 可依赖于轨迹的任意变量，但需是 γ- 折扣 Q_t 函数的无偏估计量；而 b_t 是在动作 a_t 之前采样得到的状态与动作的任意函数。

命题 1：假设优势估计量 $\hat{A}_t$ 可表示为如下形式：$\hat{A}_t(s_{0:\infty},a_{0:\infty}) = Q_t(s_{t:\infty},a_{t:\infty}) - b_t(s_{0:t},a_{0:t-1})$ 其中，对于所有 (s_t,a_t)，满足以下条件：$\mathbb{E}\left[Q_t(s_{t:\infty},a_{t:\infty}) \mid s_t,a_t\right] = Q^{\pi,\gamma}(s_t,a_t)$ 则 $\hat{A}_t$ 是 γ- 公正估计器。

该命题的证明过程详见附录 B。不难验证，以下表达式均为适用于 $\hat{A}_t$ 的 γ- 公正估计器：

- $\sum_{l=0}^{\infty}\gamma^l r_{t+l}$
- $Q^{\pi,\gamma}(s_t,a_t)$
- $A^{\pi,\gamma}(s_t,a_t)$
- $r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)$.

3 优势函数估计

本节旨在对折扣优势函数 $A^{\pi,\gamma}(s_t,a_t)$ 构建精确的估计量 $\hat{A}_t$，该估计量随后将用于构造如下形式的策略梯度估计器：

$$\hat{g}=\frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty}\hat{A}_t^n\nabla_\theta\log\pi_\theta\left(a_t^n\mid s_t^n\right) \tag{9}$$

其中，n 表示批量中的一个轨迹（episode）索引。

设 V 为近似值函数。定义 $\delta_t^V=r_t+\gamma V(s_{t+1})-V(s_t)$，即具有折扣因子 γ 的值函数 V 的时序差分（Sutton & Barto, 1998）。需注意，δ_t^V 可视为对动作 a_t 优势函数的一种估计。实际上，若值函数 V 为真实值函数（即 $V=V^{\pi,\gamma}$），则 δ_t^V 不仅是 γ-公正优势估计量，并且是 $A^{\pi,\gamma}$ 的无偏估计量，推导过程如下：

$$\begin{aligned}\mathbb{E}_{s_{t+1}}\left[\delta_t^{V^{\pi,\gamma}}\right]&=\mathbb{E}_{s_{t+1}}\left[r_t+\gamma V^{\pi,\gamma}\left(s_{t+1}\right)-V^{\pi,\gamma}\left(s_t\right)\right]\\&=\mathbb{E}_{s_{t+1}}\left[Q^{\pi,\gamma}\left(s_t,a_t\right)-V^{\pi,\gamma}\left(s_t\right)\right]=A^{\pi,\gamma}\left(s_t,a_t\right).\end{aligned} \tag{10}$$

补充：

$$\begin{aligned}Q^{\pi,\gamma}(s_t,a_t)&=\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\mid s_t,a_t\right]\\&=\mathbb{E}\left[r_t+\sum_{k=1}^{\infty}\gamma^k r_{t+k}\mid s_t,a_t\right]\\&=\mathbb{E}\left[r_t+\gamma\,\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\mid s_{t+1},s_t,\ldots s_0,a_t,\ldots a_0\right]\middle|\,s_t,a_t\right]\\&=\mathbb{E}\left[r_t+\gamma\,\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\mid s_{t+1}\right]\middle|\,s_t,a_t\right]\\&=\mathbb{E}\left[r_t+\gamma\,V_{s_{t+1}}\middle|\,s_t,a_t\right]\end{aligned}$$

这里，第三个等号用到了条件期望的性质，四个等号用到了马氏性。

然而，该估计量仅在 $V=V^{\pi,\gamma}$ 时才是 γ-公正估计量（γ-just）；否则，它将导致有偏差的策略梯度估计结果。

接下来，我们考虑对 k 个 δ 项求和，将其记为 $\hat{A}_t^{(k)}$，具体形式如下：

$$\hat{A}_t^{(1)}:=\delta_t^V\quad=-V\left(s_t\right)+r_t+\gamma V\left(s_{t+1}\right) \tag{11}$$

$$\hat{A}_t^{(2)} := \delta_t^V + \gamma\delta_{t+1}^V = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \tag{12}$$

$$\hat{A}_t^{(3)} := \delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3}) \tag{13}$$

$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) \tag{14}$$

这些等式由望远镜求和（telescoping sum）推导得出，我们可以看到，$\hat{A}_t^{(k)}$ 包含对 k 步回报的估计，再减去一个基线项 $-V(s_t)$。类似地，单步残差 δ_t^V 可表示为 $\hat{A}_t^{(1)} = \delta_t^V$。因此，$\hat{A}_t^{(k)}$ 可视为优势函数的估计器，该估计量仅在 $V = V^{\pi,\gamma}$ 时才是 γ-公正估计器。需注意，随着 $k \to \infty$，偏差通常会逐渐减小——这是因为 $\gamma^k V(s_{t+k})$ 项的折扣程度会越来越大，而 $-V(s_t)$ 项并不会对偏差产生影响。当 $k \to \infty$ 时，我们可得：

$$\hat{A}_t^{(\infty)} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V = -V(s_t) + \sum_{l=0}^{\infty} \gamma^l r_{t+l} \tag{15}$$

该式本质上是经验回报减去价值函数基线。

广义优势估计量 $GAE(\gamma,\lambda)$ 定义为上述 k 步估计量的指数加权平均，具体形式如下：

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} :=& (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots\right) \\
=& (1-\lambda)\left(\delta_t^V + \lambda\left(\delta_t^V + \gamma\delta_{t+1}^V\right) + \lambda^2\left(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V\right) + \cdots\right) \\
=& (1-\lambda)\left(\delta_t^V\left(1 + \lambda + \lambda^2 + \cdots\right) + \gamma\delta_{t+1}^V\left(\lambda + \lambda^2 + \lambda^3 + \cdots\right)\right. \\
& \left. + \gamma^2\delta_{t+2}^V\left(\lambda^2 + \lambda^3 + \lambda^4 + \cdots\right) + \cdots\right) \\
=& (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}^V\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}^V\left(\frac{\lambda^2}{1-\lambda}\right) + \cdots\right) \\
=& \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned} \tag{16}$$

由公式（16）可知，GAE 的公式实质上是一系列 Bellman 残差项的折扣加权和，具有简单而直观的形式。第 4 节进一步从奖励函数修改下的马尔可夫决策过程（MDP）视角对其进行了详细解释。我们上文所采用的构造方法，与用于定义时序差分算法 $TD(\lambda)$（Sutton & Barto, 1998）的构造方法高度相似；但需注意，$TD(\lambda)$ 是值函数的估计器，而 GAE 是优势函数的估计器。

该公式有两个值得注意的特殊情况，可通过设置 $\lambda = 0$ 和 $\lambda = 1$ 得到：

$$\text{GAE}(\gamma, 0) : \hat{A}_t := \delta_t \quad = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{17}$$

$$\text{GAE}(\gamma, 1) : \hat{A}_t := \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t) \tag{18}$$

无论价值函数 V 的精度如何，$\text{GAE}(\gamma, 1)$ 始终是 γ-公正的，但由于涉及长期回报之和，其方差较高。而 $\text{GAE}(\gamma, 0)$ 仅在 $V = V^{\pi,\gamma}$ 时才是 γ-公正的，否则会引入偏差，但具有更低方差。对于当 $0 < \lambda < 1$ 时，GAE 在偏差与方差之间提供权衡，其通过参数 λ 做出权衡。

我们提出的优势估计量包含两个独立参数 γ 和 λ，在使用近似值函数时，这两个参数均会对偏差-方差权衡产生影响。但二者的作用不同，且适用的取值范围也存在差异。其中，γ 的核心作用是决定值函数 $V^{\pi,\gamma}$ 的尺度（该尺度与 λ 无关）。若 $\gamma < 1$，则无论值函数是否准确，策略梯度估计都存在偏差。与之不同的是，仅当价值函数存在精度误差时，取 $\lambda < 1$ 才会引入偏差。从实验结果来看，我们发现 λ 的最优取值远小于 γ 的最优取值，这很可能是因为在价值函数精度尚可的情况下，λ 引入的偏差远小于 γ。

利用广义优势估计量，我们可构造折扣策略梯度 g^γ 的有偏估计量，具体形式如下：

$$g^\gamma \approx \mathbb{E}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t^{\text{GAE}(\gamma,\lambda)}\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V\right], \tag{19}$$

其中，当 $\lambda = 1$ 时，等式成立，即估计量无偏。

4 从奖励塑形角度的解读

在本节中，我们将探讨如何将 λ 解释为在对马尔可夫决策过程（MDP）进行奖励塑形转换后应用的额外折扣因子。同时，我们还将引入响应函数的概念，以帮助理解由 γ 和 λ 所引入的偏差。

奖励塑形（Reward Shaping）（Ng 等人，1999）指对 MDP 的奖励函数进行如下变换：设 $\Phi : S \to \mathbb{R}$ 为状态空间 S 上任意一个标量值函数，定义转换后的奖励函数 $\tilde{r}$ 为：

$$\tilde{r}(s, a, s') = r(s, a, s') + \gamma\Phi(s') - \Phi(s), \tag{20}$$

这一变换进而定义了一个转换后的 MDP。对于任意策略 π 而言，该变换均不会改

变折扣优势函数 $A^{\pi,\gamma}$ 的取值。为了理解这一点，考虑从状态 s_t 开始的轨迹的折扣奖励和：

$$\sum_{l=0}^{\infty}\gamma^l\tilde{r}\left(s_{t+l},a_{t+l},s_{t+l+1}\right)=\sum_{l=0}^{\infty}\gamma^l r\left(s_{t+l},a_{t+l},s_{t+l+1}\right)-\Phi\left(s_t\right). \tag{21}$$

设 $\tilde{Q}^{\pi,\gamma}$、$\tilde{V}^{\pi,\gamma}$、$\tilde{A}^{\pi,\gamma}$ 分别为转换后 MDP 的值函数与优势函数，根据这些函数的定义可推导出：

$$\tilde{Q}^{\pi,\gamma}(s,a)=Q^{\pi,\gamma}(s,a)-\Phi(s) \tag{22}$$

$$\tilde{V}^{\pi,\gamma}(s)=V^{\pi,\gamma}(s)-\Phi(s) \tag{23}$$

$$\tilde{A}^{\pi,\gamma}(s,a)=(Q^{\pi,\gamma}(s,a)-\Phi(s))-(V^{\pi,\gamma}(s)-\Phi(s))=A^{\pi,\gamma}(s,a) \tag{24}$$

需注意，若 Φ 恰好等于原始 MDP 中的状态值函数 $V^{\pi,\gamma}$，则转换后的 MDP 会具备一个有趣的特性：所有状态下的 $\tilde{V}^{\pi,\gamma}(s)$ 均为 0。

Ng 等（1999）证明了，当我们的目标是最大化折扣奖励和 $\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t,s_{t+1})$ 时，奖励塑形转换不会改变策略梯度和最优策略。相反，本文关注的是最大化未折扣奖励和的情况，其中折扣因子 γ 被用作方差缩减参数。

在回顾了奖励塑形的相关概念后，考虑如何利用它来获得策略梯度估计。最自然的方法是构造使用折扣的塑形奖励 $\tilde{r}$ 的策略梯度估计器。然而，方程（21）显示，我们得到的是原始 MDP 奖励的折扣和减去一个基线项。接下来，我们考虑使用“更陡峭的折扣因子”$\gamma\lambda$（其中 $0\leq\lambda\leq 1$）。不难发现，当我们令塑形函数 $\Phi=V$ 时，塑造后奖励 $\tilde{r}$ 恰好等于第 3 节定义的贝尔曼残差项 δ^V。令 $\Phi=V$，可得：

$$\sum_{l=0}^{\infty}(\gamma\lambda)^l\tilde{r}\left(s_{t+l},a_t,s_{t+l+1}\right)=\sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^{V}=\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)}. \tag{25}$$

因此，通过考虑折扣的塑形奖励和 $\gamma\lambda$，我们恰好得到了第 3 节中的广义优势估计器。如前文所示，当 $\lambda=1$ 时，该估计量能对折扣策略梯度 g^γ 进行无偏估计；而当 $\lambda<1$ 时，得到的则是有偏估计量。

为进一步分析该塑造变换及参数 γ 与 λ 的影响，引入“响应函数”χ 的概念是有用的，我们对其定义如下：

$$\chi\left(l;s_t,a_t\right)=\mathbb{E}\left[r_{t+l}|s_t,a_t\right]-\mathbb{E}\left[r_{t+l}|s_t\right]. \tag{26}$$

需注意，$A^{\pi,\gamma}(s,a)=\sum_{l=0}^{\infty}\gamma^l\chi(l;s,a)$，即响应函数可将优势函数按时间步分解。响应函数能够帮助我们量化时序信用分配问题：动作与奖励之间的长程依赖关系，对应于当 $l\gg 0$ 时响应函数取值非零的情况。

接下来，让我们回顾折扣因子 γ 以及我们通过使用 $A^{\pi,\gamma}$ 而非 $A^{\pi,1}$ 所做的近似。式 (6) 中的折扣策略梯度估计器具有如下形式的项之和：

$$\nabla_\theta \log \pi_\theta(a_t \mid s_t) A^{\pi,\gamma}(s_t, a_t) = \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{l=0}^{\infty} \gamma^l \chi(l; s_t, a_t). \tag{27}$$

使用折扣因子 $\gamma < 1$，等价于去掉了 $l \gg 1/(1-\gamma)$ 的项。因此，若响应函数 χ 随 l 增大而快速衰减（也就是说，若一个动作对奖励的影响在约 $1/(1-\gamma)$ 个时间步后“被遗忘”），则该近似处理所引入的误差会很小。

若奖励函数 $\tilde{r}$ 是通过 $\Phi = V^{\pi,\gamma}$ 得到的，则对于 $l > 0$，有 $\mathbb{E}[\tilde{r}_{t+l} \mid s_t, a_t] = \mathbb{E}[\tilde{r}_{t+l} \mid s_t] = 0$，即响应函数仅在 $l = 0$ 处非零。因此，该奖励塑形转换会将时序延展的响应转化为即时响应。鉴于 $V^{\pi,\gamma}$ 能完全减少了响应函数的时序跨度，我们可以希望一个好的近似 $V \approx V^{\pi,\gamma}$ 会部分减少这一时间跨度。这一观察结果为公式（16）提供了一种解读：利用 V 对奖励进行塑造，以缩小响应函数的时序范围，随后引入“更陡峭的”折扣 $\gamma\lambda$ 来截断由长延迟引发的噪声，即忽略满足 $l \gg 1/(1-\gamma\lambda)$ 的项 $\nabla_\theta \log \pi_\theta(a_t \mid s_t)\delta^{\tilde{V}}_{t+l}$。

5 值函数估计

在强化学习中，存在多种方法可用于估计值函数（参见 Bertsekas，2012）。当使用非线性函数逼近器表示值函数时，最简单的方法是求解一个非线性回归问题：

$$\underset{\phi}{\text{minimize}} \sum_{n=1}^{N} \left\| V_\phi(s_n) - \hat{V}_n \right\|^2, \tag{28}$$

其中，$\hat{V}_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$ 表示奖励的折扣和，n 为一批轨迹中所有时间步的索引。这种方法有时被称为用于价值函数估计的蒙特卡洛（Monte Carlo）方法或时序差分（TD(1)）方法。

在本文的实验中，我们使用信赖域方法来优化值函数。信赖域的引入有助于防止模型过拟合最近一次采样批次的数据。为构建信赖域问题，我们首先计算 $\sigma^2 = \frac{1}{N}\sum_{n=1}^{N} \left\| V_{\phi_{\text{old}}}(s_n) - \hat{V}_n \right\|^2$，其中 ϕ_{old} 为优化前的参数向量。随后，我们求解如下带约束的优化问题：

$$\begin{aligned} \underset{\phi}{\text{minimize}} \quad & \textstyle\sum_{n=1}^{N} \left\| V_\phi(s_n) - \hat{V}_n \right\|^2 \\ \text{subject to} \quad & \textstyle\frac{1}{N}\sum_{n=1}^{N} \frac{\left\| V_\phi(s_n) - V_{\phi_{\text{old}}}(s_n) \right\|^2}{2\sigma^2} \le \epsilon. \end{aligned} \tag{29}$$

补充：对于两个正态分布设

$$p = \mathcal{N}(\mu_1, \sigma_1^2), \quad q = \mathcal{N}(\mu_2, \sigma_2^2)$$

则

$$D_{\mathrm{KL}}(p \parallel q) = \ln \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}.$$

这里 $p_{\text{old}} = \mathcal{N}(V_{\phi_{\text{old}}}, \sigma^2)$，$q = \mathcal{N}(V_\phi, \sigma^2)$，所以约束条件等价于平均 KL 散度。

该约束等价于将前一个价值函数与新价值函数之间的平均 KL 散度（KL divergence）限制在 ϵ 以内，其中值函数被解释为具有均值 $V_\phi(s)$，方差为 σ^2 的条件高斯分布的参数化形式。

我们采用共轭梯度算法（Wright & Nocedal, 1999）求解信赖域问题的近似解。具体而言，我们需要求解如下二次规划问题：

$$\begin{aligned} &\underset{\phi}{\text{minimize}}\ g^T (\phi - \phi_{\text{old}}) \\ &\text{subject to}\ \tfrac{1}{N} \textstyle\sum_{n=1}^{N} (\phi - \phi_{\text{old}})^T H (\phi - \phi_{\text{old}}) \le \epsilon. \end{aligned} \tag{30}$$

其中，g 为目标函数的梯度，$H = \frac{1}{N} \sum_n j_n j_n^T$（$j_n = \nabla_\phi V_\phi(s_n)$）。需注意，$H$ 是目标函数海森矩阵的“高斯-牛顿”近似；若将值函数视为条件概率分布，则 H（在比例因子 σ^2 意义下）相当于其 Fisher 信息矩阵。通过矩阵-向量乘积 $v \to Hv$ 实现共轭梯度算法后，我们可计算得到步长方向 $s \approx -H^{-1}g$。随后，对 s 进行缩放（即 $s \to \alpha s$），使得 $\frac{1}{2}(\alpha s)^T H(\alpha s) = \epsilon$，最终令 $\phi = \phi_{\text{old}} + \alpha s$。该过程与我们用于更新策略的过程类似，后者将在第 6 节中进一步阐述，且基于 Schulman 等人（2015）的研究成果。

6 实验

我们设计了一组实验，旨在探究以下问题：

1. 在利用广义优势估计（GAE）优化回合总奖励时，改变参数 $\lambda \in [0, 1]$ 和 $\gamma \in [0, 1]$ 会产生怎样的实证效果？

2. 是否可以结合广义优势估计与信任区域算法（用于策略和价值函数优化），来优化用于复杂控制问题的大型神经网络策略？

6.1 策略优化算法

尽管广义优势估计可与多种不同的策略梯度方法结合使用，但在本实验中，我们采用信赖域策略优化（Schulman 等人，2015）进行策略更新。TRPO 通过在每次

迭代中近似求解如下带约束的优化问题来更新策略：

$$
\begin{aligned}
&\underset{\theta}{\text{minimize}}\ L_{\theta_{old}}(\theta) \\
&\text{subject to } \overline{D}_{\mathrm{KL}}^{\theta_{old}}(\pi_{\theta_{old}}, \pi_\theta) \le \epsilon \\
&\text{其中 } L_{\theta_{old}}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \frac{\pi_\theta(a_n|s_n)}{\pi_{\theta_{old}}(a_n|s_n)} \hat{A}_n \\
&\qquad \overline{D}_{\mathrm{KL}}^{\theta_{old}}(\pi_{\theta_{old}}, \pi_\theta) = \frac{1}{N}\sum_{n=1}^{N} D_{KL}(\pi_{\theta_{old}}(\cdot \mid s_n) \parallel \pi_\theta(\cdot \mid s_n))
\end{aligned}
\tag{31}
$$

正如 Schulman 等人（2015）所述，我们通过对目标函数进行线性化、对约束条件进行二次化来近似求解该问题，由此得到方向为 $\theta - \theta_{\text{old}} \propto -F^{-1}g$ 的更新步长，其中 F 为平均 Fisher 信息矩阵，g 为策略梯度估计量。该策略更新的步长方向与自然策略梯度（Kakade，2001a）及自然演员-评论员算法（Peters & Schaal，2008）的步长方向相同，但它采用了不同的步长确定方案和数值计算程序。

由于已有研究（Schulman 等人，2015）已将 TRPO 与多种不同的策略优化算法进行了对比，因此我们不再重复这些对比实验；相反，我们将在固定基础算法（即 TRPO）的前提下，重点研究策略梯度估计量中参数 γ 和 λ 的变化所产生的影响。

为保证完整性，下文给出迭代更新策略与价值函数的完整算法：

1: 初始化策略参数 θ_0 和价值函数参数 ϕ_0。
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: 　仿真当前策略 π_{θ_i}，直至获取 N 个时间步的数据。
4: 　利用价值函数 $V = V_{\phi_i}$，计算所有时间步 $t \in \{1, 2, \ldots, N\}$ 处的时序差分误差 δ_V^t。
5: 　计算所有时间步处的优势估计量 $\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_V^{t+l}$。
6: 　通过信赖域策略优化（TRPO）更新（公式（31））计算新的策略参数 θ_{i+1}。
7: 　通过公式（30）计算新的价值函数参数 ϕ_{i+1}。
8: **end for**

需注意，策略更新 $\theta_i \to \theta_{i+1}$ 是利用价值函数 V_ϕ 进行优势估计，而非使用 $V_{\phi_{i+1}}$。若先更新价值函数，则会引入额外偏差。为理解这一点，可考虑一种极端情况：当价值函数发生过拟合，且所有时间步的贝尔曼残差 $r_t + \gamma V(s_{t+1}) - V(s_t)$ 均变为零时，策略梯度估计量将等于零。

6.2 实验设置

我们在经典的倒立摆平衡任务以及多个具有挑战性的 3D 运动控制任务上对所提方法进行了评估，这些 3D 任务包括：（1）双足行走；（2）四足行走；（3）双足机器人从平躺状态动态站立。实验所用的机器人模型如图 1 所示。

6.2.1 网络架构

对于所有 3D 机器人任务，我们采用了相同的神经网络架构——该架构为前馈神经网络，包含 3 个隐藏层，各隐藏层分别设有 100 个、50 个和 25 个 tanh 激活单元。该架构同时用于策略网络和价值函数网络。最终输出层采用线性激活函数。价值函数估计器同样使用该架构，但仅输出一个标量值（用于表示状态价值）。对于更简单的倒立摆任务，我们采用线性策略），并使用一个含 20 个单元隐藏层的神经网络作为价值函数网络。

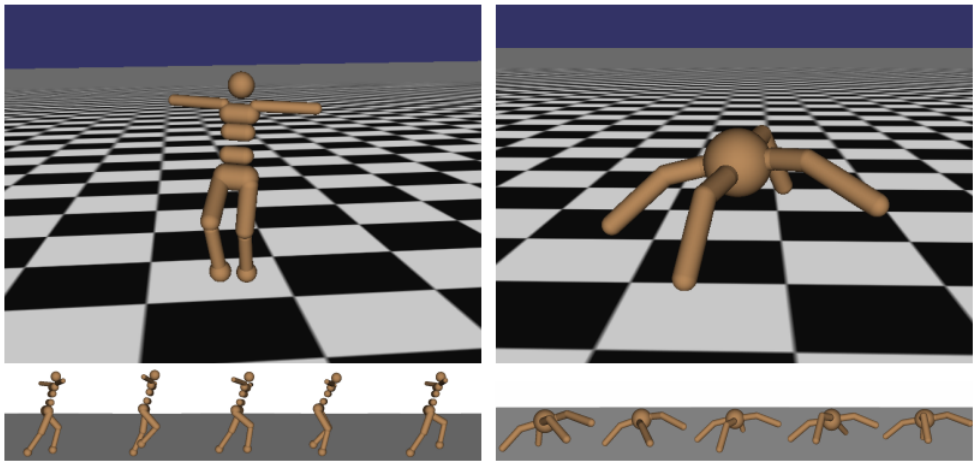


图 0.54　用于 3D 运动控制的机器人模型。下图：从习得步态中截取的一系列帧。相关视频可在链接 https://sites.google.com/site/gaepapersupp 查看。

6.2.2 任务细节

在倒立摆平衡任务中，我们每个批次收集 20 条轨迹，每条轨迹的最大长度为 1000 个时间步，所使用的物理参数参考自 Barto 等人（1983）的研究成果。

所有机器人仿真任务均通过 MuJoCo 物理引擎完成（Todorov 等人，2012）。其中，双足人形机器人模型的状态空间维度为 33，可驱动自由度为 10；四足机器人模型的状态空间维度为 29，可驱动自由度为 8。这些任务的初始状态均从以参考构型为中心的均匀分布中采样得到。在双足行走任务中，每个批次使用 50000 个时间步的数据；在四足行走和双足站立任务中，每个批次使用 200000 个时间步的数据。若机器人在 2000 个时间步内未达到终止状态，则该回合将在第 2000 个时间步时终止。仿真的时间步长设定为 0.01 秒。奖励函数见下表。此处，v_{fwd} 表示前

任务	奖励函数
3D 双足行走	$v_{\text{fwd}} - 10^{-5}\|u\|^2 - 10^{-5}\|f_{\text{impact}}\|^2 + 0.2$
四足行走	$v_{\text{fwd}} - 10^{-6}\|u\|^2 - 10^{-3}\|f_{\text{impact}}\|^2 + 0.05$
双足站立	$-(h_{\text{head}} - 1.5)^2 - 10^{-5}\|u\|^2$

进速度，u 表示关节力矩向量，f_{impact} 表示碰撞力，h_{head} 表示头部高度。

在运动控制任务中，若智能体（此处指机器人）的质心低于预设高度，该回合将终止：双足机器人的预设高度为 0.8 米，四足机器人的预设高度为 0.2 米。奖励函数中的常数项用于鼓励延长回合时长；若缺少这一常数项，奖励函数中的二次惩罚项可能会导致策略倾向于尽快结束回合。

6.3 实验结果

所有结果均以“代价”为衡量指标，代价定义为奖励的负值，实验目标为最小化代价。习得策略的相关视频可访问链接：https://sites.google.com/site/gaepapersupp 查看。在图表中，“No VF”表示我们未使用状态价值函数的估计值，而是采用了一个与状态无关的时间依赖型基线；该时间依赖型基线通过计算批次内所有轨迹在每个时间步的回报均值得到。

6.3.1 实验结果

所有结果均为 21 组不同随机种子实验的平均值。结果如图 2 所示，表明当参数取中间值时可获得最佳效果，即 $\gamma \in [0.96, 0.99]$ 且 $\lambda \in [0.92, 0.99]$。

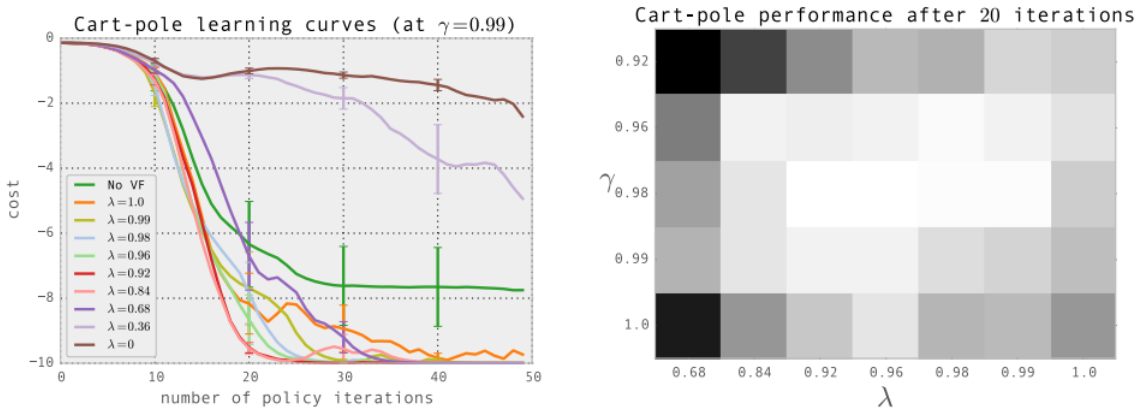


图 0.55　左图：在 $\gamma = 0.99$ 的情况下，采用广义优势估计（GAE）且 λ 取不同值时，倒立摆任务的学习曲线。当 $\lambda \in [0.92, 0.99]$ 时，策略提升速度最快。右图：随着 γ 和 λ 取值的变化，经过 20 轮策略优化后的性能。白色代表更高的奖励，当 γ 和 λ 均取中间值时，可获得最佳结果。

6.3.2 三维双足行走任务

每次实验在 16 核机器上运行约 2 小时，其中仿真轨迹生成采用了并行处理方式，策略与价值函数优化过程中所用的函数计算、梯度求解以及矩阵-向量积计算也均采用并行处理。此处的结果为 9 组不同随机种子实验的平均值。最佳性能仍出现在参数取中间值的情况下，即 $\gamma \in [0.99, 0.995]$、$\lambda \in [0.96, 0.99]$。经过 1000 轮迭代后，得到的步态速度快、动作平滑且稳定性强，基本可实现完全稳定的行走。

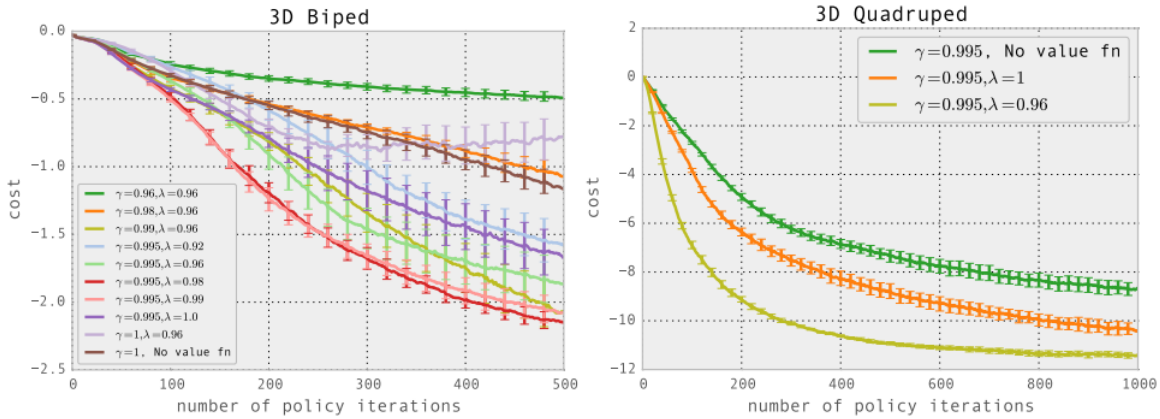


图 0.56　左图：3D 双足行走任务的学习曲线，该曲线为算法 9 次运行结果的平均值。右图：3D 四足行走任务的学习曲线，该曲线为算法 5 次运行结果的平均值。

我们可计算该学习过程所用的“真实时间”：0.01 秒/时间步 ×50000 时间步/批次 ×1000 批次 ÷（3600×24 秒/天）=5.8 天。因此，若能找到重置机器人状态的方法并确保其在实验过程中不损坏，该算法有望在真实机器人（或多台并行学习的真实机器人）上运行。

6.3.3 其他 3D 机器人任务

我们研究的另外两种运动行为分别是四足行走和三维双足机器人从地面站立起来。同样，每种实验条件下我们都进行了 5 组实验，每组实验使用不同的随机种子和初始化参数。在 32 核机器上，每组实验约耗时 4 小时。由于运行这些实验需要大量计算资源，我们在这些任务领域上进行了更有限的对比实验：固定 $\gamma = 0.995$，但将 λ 分别设为 0 和 0.96，同时还设置了一个不使用价值函数（no value function）的实验条件作为对照。对于四足行走任务，使用价值函数且 $\lambda = 0.96$ 时获得了最佳结果（参见 6.3.2 节）。对于三维站立任务，价值函数始终能起到辅助提升效果，但 $\lambda = 0.96$ 和 $\lambda = 1$ 时的结果大致相当。

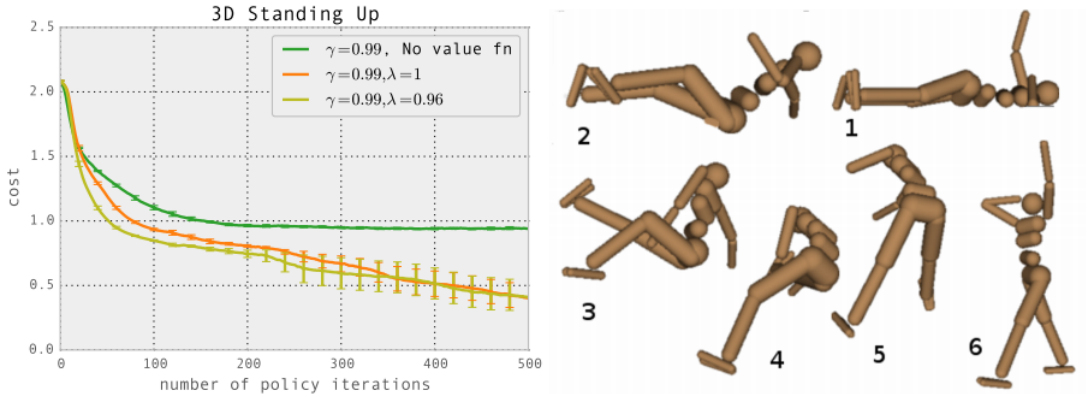


图 0.57　(a) 四足行走的学习曲线，(b) 三维站立的学习曲线，(c) 三维站立的片段。

7 讨论

策略梯度方法通过提供无偏的梯度估计，提供了一种将强化学习转化为随机梯度下降的方法。然而，到目前为止，它们在解决复杂控制问题上的成功仍然有限，主要原因是它们的高样本复杂度。我们认为，减少方差的关键是获得对优势函数的良好估计。

我们提供了一个直观但非正式的优势函数估计问题分析，并证明了广义优势估计器，该估计器有两个参数 λ 和 γ，用于调整偏差-方差的折衷。我们描述了如何将这一思想与信任区域策略优化相结合，并介绍了一种优化价值函数的信任区域算法，这两个都由神经网络表示。结合这些技术，我们能够学会解决以前对通用强化学习方法来说几乎无法解决的复杂控制任务。

我们对广义优势估计的主要实验验证是在模拟机器人运动领域。如实验所示，选择 $\lambda \in [0.9, 0.99]$ 范围内的适当中间值通常能得到最佳性能。未来工作的一个可能方向是如何以自适应或自动化的方式调整估计器参数 λ 和 γ。

另一个值得未来研究的问题是价值函数估计误差与策略梯度估计误差之间的关系。如果知道这个关系，我们就可以选择一个与关注量（通常是策略梯度估计的准确性）匹配的价值函数拟合误差度量。一些候选的误差度量包括贝尔曼误差或投影贝尔曼误差，正如 Bhatnagar 等人（2009）中所描述的那样。

另一个诱人的可能性是使用共享函数逼近架构来表示策略和价值函数，同时使用广义优势估计优化策略。虽然将这个问题以适合数值优化的方式进行表述并提供收敛性保证仍然是一个开放问题，但这种方法可能允许价值函数和策略表示共享输入的有用特征，从而实现更快的学习。

在同时进行的研究中，研究人员正在开发涉及对连续值动作进行微分的策略梯度方法（Lillicrap 等，2015；Heess 等，2015）。尽管我们通过实验发现一步回报（$\lambda = 0$）会导致过度偏差和较差的性能，但这些论文表明，当适当调优时，这些方法可以有效工作。然而，值得注意的是，这些论文考虑的控制问题具有比本文所讨论的状态和动作空间低得多的维度。对两类方法进行比较将是未来工作的一个有用方向。

A 常见问题

A.1 与兼容特征的关系是什么？

兼容特征通常与使用值函数的策略梯度算法相关联，这一概念在 Konda 和 Tsitsiklis（2003）提出的《On Actor-Critic Methods》一文中被提出。作者指出，由

于策略的表示能力有限，策略梯度仅依赖于优势函数空间的某个子空间。这个子空间是由兼容特征 $\nabla_{\theta_i} \log \pi_\theta(a_t|s_t)$ 张成的，其中 $i \in \{1, 2, ..., \dim \theta\}$。该兼容特征理论未能提供如何利用问题的时间结构来获得更好的优势函数估计的指导，因此该理论与本文所提出的理念在核心思路上基本无关。

兼容特征这一理念为计算自然策略梯度提供了一种简洁优雅的方法（Kakade, 2001a; Peters & Schaal, 2008）。给定每个时间步的优势函数的经验估计 $\hat{A}_t$，我们可以通过求解以下最小二乘问题，将其投影到兼容特征的子空间中：

$$\underset{\mathbf{r}}{\text{minimize}} \sum_t \left\| \mathbf{r} \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t) - \hat{A}_t \right\|^2. \tag{32}$$

如果 $\hat{A}$ 是 γ-公正估计器，最小二乘解就是自然策略梯度（Kakade, 2001a）。请注意，任何优势函数的估计器都可以代入此公式，包括我们在本文中推导的估计器。对于我们的实验，我们也计算了自然策略梯度步长，但我们使用了 Schulman 等人（2015）提出的更具计算效率的数值方法，正如第 6 节中所讨论的。

A.2 为何不直接使用 Q 函数？

以往的演员-评论家方法，例如 Konda 和 Tsitsiklis（2003）提出的方法，使用 Q-函数来获得潜在的低方差策略梯度估计。最近的研究论文，包括 Heess 等人（2015）和 Lillicrap 等人（2015），已经展示了神经网络 Q-函数近似器可以有效地应用于策略梯度方法。然而，使用状态值函数以本文的方法存在几个优势。首先，状态值函数的输入维度较低，因此比状态-动作值函数更易于学习。其次，本文的方法使我们能够在高偏差估计器（$\lambda = 0$）和低偏差估计器（$\lambda = 1$）之间平滑地插值。另一方面，使用参数化 Q-函数只能使我们使用高偏差估计器。我们发现，当使用一步回报估计时，即 $\lambda = 0$ 估计器时，偏差过大，$\hat{A}_t = \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$。我们预计，当使用涉及参数化 Q-函数的优势估计器时，$\hat{A}_t = Q(s, a) - V(s)$，也会遇到类似的问题。使用参数化 Q-函数并尝试减少偏差的算法空间是有趣的，但探索这些可能性超出了本文的范围。

B 证明

命题 1 的证明：首先，我们可将期望分解为包含 Q 函数和 b 函数的项，具体如下：

$$\begin{aligned} &\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(Q_t(s_{t:\infty}, a_{t:\infty}) - b_t(s_{0:t}, a_{0:t-1})\right)\right] \\ &= \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot Q_t(s_{t:\infty}, a_{t:\infty})\right] - \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot b_t(s_{0:t}, a_{0:t-1})\right] \end{aligned} \tag{33}$$

我们将依次分析含 Q 函数和 b 函数的项。

$$
\begin{aligned}
&\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) Q_t(s_{t:\infty}, a_{t:\infty})\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) Q_t(s_{t:\infty}, a_{t:\infty}) \mid s_t, a_t\right]\right] \\
&= \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \mathbb{E}\left[Q_t(s_{t:\infty}, a_{t:\infty}) \mid s_t, a_t\right]\right] \\
&= \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t)\right]
\end{aligned}
$$

接下来，

$$
\begin{aligned}
&\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot b_t(s_{0:t}, a_{0:t-1})\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot b_t(s_{0:t}, a_{0:t-1}) \mid s_{0:t}, a_{0:t-1}\right]\right] \\
&= \mathbb{E}\left[b_t(s_{0:t}, a_{0:t-1}) \cdot \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \mid s_{0:t}, a_{0:t-1}\right]\right] \\
&= \mathbb{E}\left[b_t(s_{0:t}, a_{0:t-1}) \cdot 0\right] \\
&= 0
\end{aligned}
$$

参考文献

[1] Barto, Andrew G, Sutton, Richard S, and Anderson, Charles W. Neuronlike adaptive elements that can solve difficult learning control problems. Systems, Man and Cybernetics, IEEE Transactions on, (5):834–846, 1983.

[2] Baxter, Jonathan and Bartlett, Peter L. Reinforcement learning in POMDPs via direct gradient ascent. In ICML, pp. 41–48, 2000.

[3] Bertsekas, Dimitri P. Dynamic programming and optimal control, volume 2. Athena Scientific, 2012.

[4] Bhatnagar, Shalabh, Precup, Doina, Silver, David, Sutton, Richard S, Maei, Hamid R, and Szepesv´ari, Csaba. Convergent temporal-difference learning with arbitrary smooth function approximation. In Advances in Neural Information Processing Systems, pp. 1204–1212, 2009.

[5] Greensmith, Evan, Bartlett, Peter L, and Baxter, Jonathan. Variance reduction techniques for gradient estimates in reinforcement learning. The Journal of Machine Learning Research, 5:1471–1530, 2004.

[6] Hafner, Roland and Riedmiller, Martin. Reinforcement learning in feedback control. Machine learning, 84 (1-2):137–169, 2011.

[7] Heess, Nicolas, Wayne, Greg, Silver, David, Lillicrap, Timothy, Tassa, Yuval, and Erez, Tom. Learning continuous control policies by stochastic value gradients. arXiv preprint arXiv:1510.09142, 2015.

[8] Hull, Clark. Principles of behavior. 1943.

[9] Kakade, Sham. A natural policy gradient. In NIPS, volume 14, pp. 1531–1538, 2001a.

[10] Kakade, Sham. Optimizing average reward using discounted rewards. In Computational Learning Theory, pp. 605–615. Springer, 2001b.

[11] Kimura, Hajime and Kobayashi, Shigenobu. An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function. In ICML, pp. 278–286, 1998.

[12] Konda, Vijay R and Tsitsiklis, John N. On actor-critic algorithms. SIAM journal on Control and Optimization, 42(4):1143–1166, 2003.

[13] Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[14] Marbach, Peter and Tsitsiklis, John N. Approximate gradient methods in policy-space optimization of markov reward processes. Discrete Event Dynamic Systems, 13(1-2):111–148, 2003.

[15] Marbach, Peter and Tsitsiklis, John N. Approximate gradient methods in policy-space optimization of markov reward processes. Discrete Event Dynamic Systems, 13(1-2):111–148, 2003.

[16] Minsky, Marvin. Steps toward artificial intelligence. Proceedings of the IRE, 49(1):8–30, 1961.

[17] Ng, Andrew Y, Harada, Daishi, and Russell, Stuart. Policy invariance under reward transformations: Theory and application to reward shaping. In ICML, volume 99, pp. 278–287, 1999.

[18] Peters, Jan and Schaal, Stefan. Natural actor-critic. Neurocomputing, 71(7):1180–1190, 2008.

[19] Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I, and Abbeel, Pieter. Trust region policy optimization. arXiv preprint arXiv:1502.05477, 2015.

[20] Sutton, Richard S and Barto, Andrew G. Introduction to reinforcement learning. MIT Press, 1998.

[21] Sutton, Richard S, McAllester, David A, Singh, Satinder P, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In NIPS, volume 99, pp. 1057–1063. Citeseer, 1999.

[22] Thomas, Philip. Bias in natural actor-critic algorithms. In Proceedings of The 31st International Conference on Machine Learning, pp. 441–448, 2014.

[23] Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 5026–5033. IEEE, 2012.

[24] Wawrzy´nski, Paweł. Real-time reinforcement learning by sequential actor–critics and experience replay. Neural Networks, 22(10):1484–1497, 2009.

[25] Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229–256, 1992.

[26] Wright, Stephen J and Nocedal, Jorge. Numerical optimization. Springer New York, 1999.

近端策略优化算法[1,2]

摘要

我们提出了一类用于强化学习的新型策略梯度方法，该方法在通过与环境交互采样数据和使用随机梯度上升优化“代理”目标函数之间交替进行。标准的策略梯度方法每个数据样本执行一次梯度更新，而我们提出了一种新颖的目标函数，它支持小批量更新的多个轮次。这种被我们称为近端策略优化（PPO）的新方法，具备信赖域策略优化（TRPO）的部分优势，但实现起来要简单得多，更具通用性，并且样本复杂度（经验上）更优。我们的实验在一系列基准任务上测试了 PPO，包括模拟的机器人运动和雅达利（Atari）游戏，结果表明 PPO 优于其他在线策略梯度方法，并且在样本复杂度、简便性和实际运行时间之间总体上达到了良好的平衡。

1 引言

近年来，人们提出了几种不同的使用神经网络函数逼近器的强化学习方法。主要的竞争者有深度 Q - 学习[Mni+15]、“vanilla”策略梯度方法（[Mni + 16]）以及信赖域/自然策略梯度方法[Sch+15b]。然而，在开发一种可扩展（针对大型模型和并行实现）、数据高效且鲁棒（即，无需超参数调优就能在各种问题上成功）的方法方面，仍有改进的空间。Q - 学习（带函数逼近）在许多简单问题上都不奏效[3]，且人们对其了解甚少；vanilla 策略梯度方法的数据效率和鲁棒性都很差；而信赖域策略优化（TRPO）相对复杂，并且与包含噪声（如 dropout）或参数共享（在策略和价值函数之间，或与辅助任务之间）的架构不兼容。

本文旨在通过引入一种仅使用一阶优化就能达到 TRPO 的数据效率和可靠性能的算法，来改善当前的状况。我们提出了一种带有截断概率比的新颖目标函数，它对策略的性能形成了一个悲观估计（即下限）。为了优化策略，我们在从策略中采样数据和对采样数据进行若干轮优化之间交替进行。

---

[1]原文：Schulman J, Wolski F, Dhariwal P, et al.Proximal Policy Optimization Algorithms[J]. 2017.DOI:10.48550/arXiv.1707.06347.

[2]译者：原增昀、张晨曦、袁梓凌。在原文基础上增补了部分推导细节并提供了补充材料辅助理解，文中图片均为原文截屏。

[3]虽然 DQN 在像街机学习环境（[Bel + 15]）这样具有离散动作空间的游戏环境中运行良好，但它在诸如 OpenAI Gym（[Bro + 16]）以及 Duan 等人（[Dua + 16]）所描述的连续控制基准测试中，尚未被证明能有良好的表现。

我们的实验比较了替代目标函数的各种不同版本的性能，发现带有截断概率比的版本表现最佳。我们还将 PPO 与文献中的几种先前算法进行了比较。在连续控制任务上，它比我们所对比的几种算法表现更好。在雅达利（Atari）游戏上，它的表现（在样本复杂度方面）显著优于 A2C，并且与 ACER 相当，尽管它要简单得多。

2 背景：策略优化

2.1 策略梯度方法

策略梯度方法通过计算策略梯度的估计量，并将其代入随机梯度上升算法来工作。最常用的梯度估计量形式为：

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right] \tag{1}$$

其中，π_θ 是一个随机策略，$\hat{A}_t$ 是时间步 t 处优势函数的估计量。这里，期望 $\hat{\mathbb{E}}_t[\ldots]$ 表示在一个在采样和优化之间交替进行的算法中，对有限批次样本的经验平均。使用自动微分软件的实现方式是通过构造一个目标函数，其梯度就是策略梯度估计量；估计量 $\hat{g}$ 是通过对目标函数求导得到的：

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]. \tag{2}$$

虽然使用同一段轨迹对该损失 L^{PG} 进行多步优化很有吸引力，但这样做并没有充分的理论依据，而且从经验上看，这往往会导致破坏性的大幅策略更新（见第 6.1 节；相关结果未展示，但与“无截断或惩罚”设置的结果类似或更差）。

2.2 信赖域方法

在 TRPO[Sch+15b] 中，一个目标函数（“替代”目标函数）在策略更新规模受到约束的情况下被最大化。具体而言，

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \tag{3}$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right] \right] \leq \delta. \tag{4}$$

这里，θ_{old} 是更新前的策略参数向量。在对目标函数进行线性近似、对约束进行二次近似后，这个问题可以使用共轭梯度算法高效地近似求解。

TRPO 的理论依据实际上建议使用惩罚项而非约束，即求解无约束优化问题

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right] \right] \tag{5}$$

其中 β 是某个系数。这是因为某个替代目标函数（它计算的是各状态下的最大 KL 而非均值 KL）对策略 π 的性能构成了一个下限（即悲观边界）。TRPO 使用硬约束而非惩罚项，是因为很难选择一个单一的 β 值，使其在不同问题上都表现良好——甚至在单个问题中，随着学习过程的推进，问题的特性会发生变化，也难以选择这样的 β 值。因此，为了实现我们的目标，即得到一个能模拟 TRPO 单调改进特性的一阶算法，实验表明，仅仅选择一个固定的惩罚系数 β 并用 SGD 优化惩罚目标函数（式 5）是不够的；还需要额外的修改。

3 截断替代目标函数

令 $r_t(\theta)$ 表示概率比 $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$，因此 $r(\theta_{\text{old}}) = 1$。TRPO 最大化一个“替代”目标函数：

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]. \tag{6}$$

上标 CPI 指的是保守策略迭代[KL02]，该目标函数在其中被提出。如果没有约束，对 L^{CPI} 的最大化会导致过大的策略更新；因此，我们现在考虑如何修改目标函数，以惩罚使 $r_t(\theta)$ 偏离 1 的策略变化。

我们提出的主要目标函数如下：

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{7}$$

其中 ϵ 是一个超参数，比如 $\epsilon = 0.2$。该目标函数的动机如下。min 内部的第一项是 L^{CPI}。第二项 $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ 通过截断概率比来修改替代目标函数，这消除了将 r_t 移到区间 $[1-\epsilon, 1+\epsilon]$ 之外的动机。最后，我们取截断后和未截断目标函数的最小值，因此最终目标函数是未截断目标函数的下限（即悲观边界）。通过这种方案，我们仅在概率比的变化会使目标函数改善时忽略它，而在其使目标函数变差时将其纳入。注意，在 θ_{old} 附近（即 $r = 1$ 处），$L^{CLIP}(\theta)$ 与 $L^{CPI}(\theta)$ 是一阶近似的，但当 θ 远离 θ_{old} 时，它们就会变得不同。图0.58 绘制了 L^{CLIP} 中的单个项（即单个 t）；注意，概率比 r 会根据优势是正还是负，被截断在 $1-\epsilon$ 或 $1+\epsilon$ 处。

图 0.59 为替代目标函数 L^{CLIP} 提供了另一种直观理解。它展示了在一个连续控制问题上，当我们沿着由近端策略优化（我们即将介绍的算法）得到的策略更新方向进行插值时，几个目标函数是如何变化的。我们可以看到，L^{CLIP} 是 L^{CPI} 的下限，对过大的策略更新有惩罚作用。

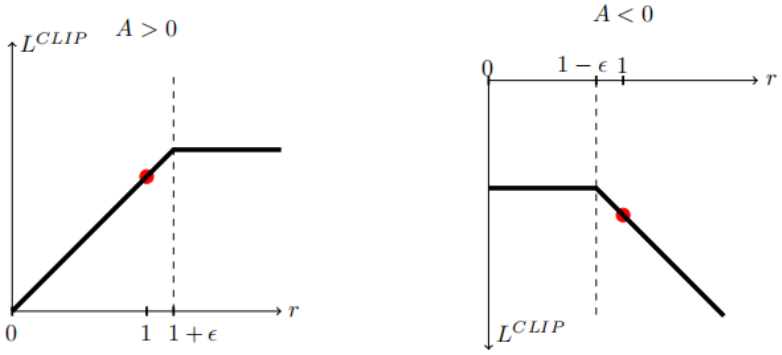


图 0.58　展示替代函数 L^{CLIP} 的单个项（即单个时间步）作为概率比 r 的函数的图像，分别对应正优势（左）和负优势（右）。每个图像上的红圈显示了优化的起点，即 $r = 1$。注意 L^{CLIP} 是这些项的总和。

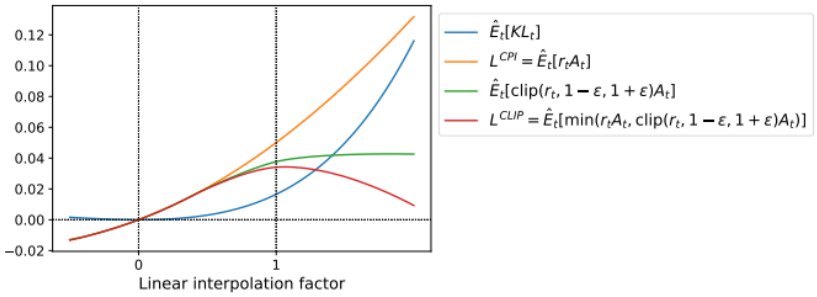


图 0.59　当我们在初始策略参数 θ_{old} 和更新后的策略参数之间进行插值时的替代目标函数。更新后的策略是我们在 PPO 的一次迭代后计算得到的，它与初始策略的 KL 散度约为 0.02，而这正是 L^{CLIP} 取得最大值的点。该图对应于在 Hopper - v1 问题上的第一次策略更新，使用的是第 6.1 节中提供的超参数。

4 自适应 KL 惩罚系数

另一种方法可以用作截断替代目标的替代方案，或者与之一同使用，即对 KL 散度采用惩罚，并自适应调整惩罚系数，以在每次策略更新时达到 KL 散度的某个目标值 d_{target}。在我们的实验中，发现 KL 惩罚的表现比截断替代目标差，但我们还是将其包含在此处，因为它是一个重要的基准。

在该算法最简单的实现中，我们在每次策略更新时执行以下步骤：

- 使用小批量 SGD 的若干轮次，优化带 KL 惩罚的目标函数

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \text{KL}\left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)\right] \right] \tag{8}$$

- 计算 $d = \hat{\mathbb{E}}_t \left[\text{KL}\left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)\right]\right]$
  - 如果 $d < d_{\text{target}}/1.5$，则 $\beta \leftarrow \beta/2$
  - 如果 $d > d_{\text{target}} \times 1.5$，则 $\beta \leftarrow \beta \times 2$

更新后的 β 用于下一次策略更新。通过这种方案，我们偶尔会看到 KL 散度

与 d_{target} 显著不同的策略更新，但这种情况很少见，且 β 会快速调整。上面的参数 1.5 和 2 是凭经验选择的，但算法对它们不是很敏感。β 的初始值是另一个超参数，但在实践中并不重要，因为算法会快速调整它。

5 算法

前几节中的替代损失可以通过对典型策略梯度实现进行微小改动来计算和求导。对于使用自动微分的实现，只需构造损失 L^{CLIP} 或 L^{KLPEN} 来替代 L^{PG}，然后对该目标函数执行多步随机梯度上升。

大多数计算方差降低的优势函数估计的技术都会利用学习到的状态价值函数 $V(s)$；例如，广义优势估计[Sch+15a]，或者[Mni+16] 中的有限 horizon 估计器。如果使用在策略和价值函数之间共享参数的神经网络架构，我们必须使用一个结合策略替代项和价值函数误差项的损失函数。如过去的工作[Mni+16,Wil92] 所建议的，该目标还可以通过添加熵奖励来进一步增强，以确保充分的探索。结合这些项，我们得到以下目标函数，它在每次迭代中（近似地）被最大化：

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{9}$$

其中 c_1, c_2 是系数，S 表示熵奖励，L_t^{VF} 是平方误差损失 $(V_\theta(s_t) - V_t^{\text{target}})^2$。更详细的说明请见附录 。

一种策略梯度实现风格在[Mni+16] 中得到推广，并且非常适合与循环神经网络一起使用，这种风格会让策略运行 T 个时间步（其中 T 远小于回合长度），并使用收集到的样本来进行更新。这种风格需要一个不考虑时间步 T 之后情况的优势估计器。[Mni+16] 使用的估计器是：

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t-1} r_{T-1} + \gamma^{T-t} V(s_T) \tag{10}$$

其中 t 指定在给定的长度为 T 的轨迹段中 $[0, T]$ 内的时间索引。对这一选择进行推广，我们可以使用广义优势估计的截断版本，当 $\lambda = 1$ 时，它就简化为式 10：

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{11}$$

其中

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{12}$$

下面展示了使用固定长度轨迹段的近端策略优化（PPO）算法。每次迭代时，N 个（并行的）智能体中的每一个都会收集 T 个时间步的数据。然后，我们基于这些 NT 个时间步的数据构造替代损失，并使用小批量 SGD（或者通常为了获得更好的性能，使用 Adam[KB14] 对其进行 K 轮优化。

算法 8 PPO，Actor - Critic 风格

for iteration = 1, 2, … **do**
  for actor = 1, 2, …, N **do**
    在环境中运行策略 $\pi_{\theta_{\text{old}}}$ 持续 T 个时间步
    计算优势估计 $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  相对于 θ 优化替代损失 L，使用 K 轮迭代且小批量大小 $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

6 实验

6.1 替代目标函数的比较

首先，我们在不同超参数下比较几种不同的替代目标函数。在此，我们将替代目标函数 L^{CLIP} 与几种自然变体和消融版本进行比较。

- 无截断或惩罚：$L_t(\theta) = r_t(\theta)\hat{A}_t$
- 截断：$L_t(\theta) = \min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)$
- KL 惩罚（固定或自适应）：$L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]$

对于 KL 惩罚，可以使用固定的惩罚系数 β，或者使用第 4 节中描述的、基于目标 KL 值 d_{target} 的自适应系数。需要注意的是，我们也尝试了在对数空间中进行截断，但发现性能并没有更好。

由于我们要在超参数上进行搜索，所以选择了计算成本低的基准测试来测试各种算法变体。具体来说，我们使用了 OpenAI Gym[Bro+16] 中实现的 7 个模拟机器人任务[4]，这些任务使用 MuJoCo[TET12] 物理引擎。我们在每个任务上进行了 100 万时间步的训练。除了用于截断的超参数（ϵ）以及我们要搜索的 KL 惩罚超参数（β, d_{target}）之外，其他超参数如表 3 所示。

为了表示策略，我们使用了一个具有两个隐藏层（每层 64 个单元）的全连接多层感知机（MLP），并使用双曲正切函数来处理非线性，输出高斯分布的均值，且标准差可变，这遵循了[Sch+15b,Dua+16] 的方法。我们不在策略和价值函数之间共享参数（因此系数 c_1 无关紧要），也不使用熵奖励。

每种算法都在所有 7 个环境上运行，每个环境使用 3 个随机种子。我们通过计算最后 100 个回合的平均总奖励来为算法的每次运行评分。我们对每个环境的分数进行了平移和缩放，使得随机策略的得分为 0，最佳结果的得分为 1，然后对 21 次运行取平均，为每种算法设置生成一个标量值。

[4] 半猎豹（HalfCheetah）、跳虫（Hopper）、倒立双摆（InvertedDoublePendulum）、倒立摆（InvertedPendulum）、伸手器（Reacher）、游泳者（Swimmer）和 Walker2d，均为“-v1”版本。

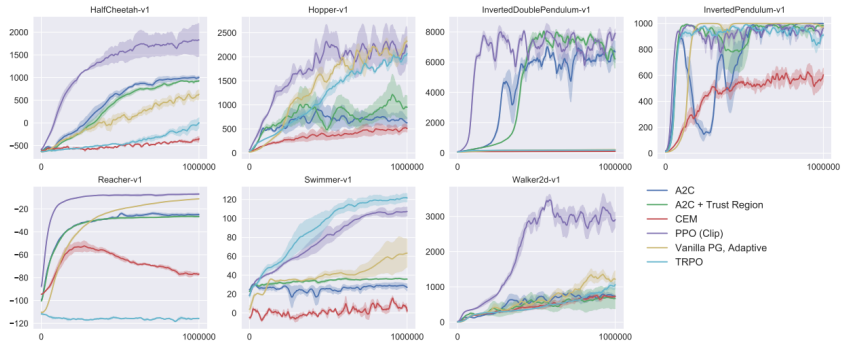


图 0.60　几种算法在多个 MuJoCo 环境上的比较，训练时长为 100 万时间步。

结果如表 24 所示。需要注意的是，对于无截断或惩罚的设置，分数为负，因为在一个环境（半猎豹）中，它会导致非常负的分数，比初始随机策略更差。

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{target}} = 0.003$	0.68
Adaptive KL $d_{\text{target}} = 0.01$	0.74
Adaptive KL $d_{\text{target}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

表 24　连续控制基准测试的结果。每种算法/超参数设置的平均标准化分数（在 7 个环境上，算法运行 21 次）。β 初始化为 1。

6.2 与连续域中其他算法的比较

接下来，我们将 PPO（使用第 3 节中的“截断”替代目标函数）与文献中几种被认为对连续问题有效的其他方法进行比较。我们对比了以下算法的调优实现：信赖域策略优化[Sch+15b]、交叉熵方法（CEM）[SL06]、带有自适应步长的 vanilla 策略梯度[5]、A2C[Mni+16]、带有信赖域的 A2C[Wan+16]。A2C 代表优势 Actor - Critic，是 A3C 的同步版本，我们发现它的性能与异步版本相同或更优。对于 PPO，我们使用了前一节中的超参数，其中 $\epsilon = 0.2$。我们发现，在几乎所有连续控制环境中，PPO 都优于之前的方法。

[5]在每批数据之后，基于原始策略和更新后策略的 KL 散度，使用与第 4 节中所示类似的规则来调整 Adam 步长。相关实现可在 https://github.com/berkeleydeeprlcourse/homework/tree/master/hw4 获取。

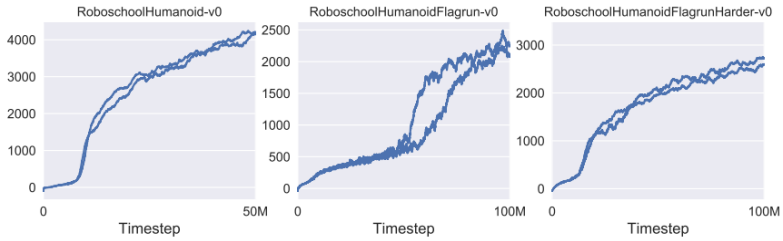


图 0.61　PPO 在使用 Roboschool 的 3D 类人机器人控制任务上的学习曲线。

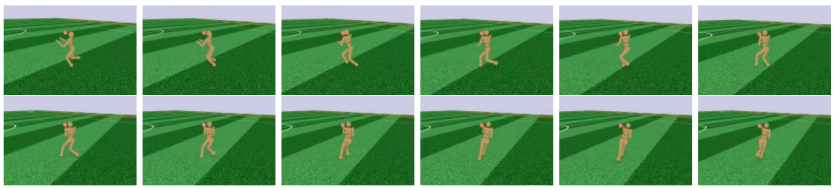


图 0.62　从 RoboschoolHumanoidFlagrun 学习到的策略的静态帧。在前六帧中，机器人朝一个目标奔跑。然后目标位置被随机改变，机器人转向并朝新目标奔跑。

6.3 连续域展示：类人机器人奔跑与转向

为了展示 PPO 在高维连续控制问题上的性能，我们在一组涉及 3D 类人机器人的问题上进行训练，机器人需要奔跑、转向，还可能在被立方体投掷时从地面起身。我们测试的三个任务是：（1）RoboschoolHumanoid：仅向前移动；（2）RoboschoolHumanoidFlagrun：目标位置每 200 个时间步或在达到目标时随机变化；（3）RoboschoolHumanoidFlagrunHarder：机器人会被立方体投掷，且需要从地面起身。学习到的策略的静态帧见图0.62，三个任务的学习曲线见图0.61。超参数见表4。在并行工作中，Heess 等人[Hee+17] 使用 PPO 的自适应 KL 变体（第 4 节）来学习 3D 机器人的移动策略。

6.4 在雅达利（Atari）领域与其他算法的比较

我们还在街机学习环境[Bel+15] 基准测试上运行了 PPO，并与 A2C[Mni+16] 和 ACER[Wan+16] 的精心调优实现进行了比较。对于这三种算法，我们使用了与[Mni+16] 中相同的策略网络架构。PPO 的超参数见表6。对于另外两种算法，我们使用了为在该基准测试上最大化性能而调优的超参数。

所有 49 款游戏的结果表和学习曲线见附录 B。我们考虑以下两种评分指标：（1）整个训练期间每回合的平均奖励（有利于快速学习）；（2）训练最后 100 回合每回合的平均奖励（有利于最终性能）。表 25 显示了每种算法“获胜”的游戏数量，其中我们通过对三次试验的评分指标取平均来确定胜者。

	A2C	ACER	PPO	Tie
(1) 整个训练期间每回合的平均奖励	1	18	**30**	0
(2) 训练最后 100 回合每回合的平均奖励	1	**28**	19	1

表 25　每种算法“获胜”的游戏数量，其中评分指标是对三次试验取平均后的结果。

7 结论

我们介绍了近端策略优化（Proximal Policy Optimization），这是一类策略优化方法，每一次策略更新都使用多轮随机梯度上升。这些方法具备信赖域方法的稳定性和可靠性，但实现起来要简单得多，只需对 vanilla 策略梯度实现修改几行代码即可，适用于更通用的场景（例如，在策略和价值函数采用联合架构时），并且整体性能更优。

8 致谢

感谢 OpenAI 的 Rocky Duan、Peter Chen 以及其他人员提出的富有见地的意见。

参考文献

[Bel+15] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. “The arcade learning environment: An evaluation platform for general agents”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[Bro+16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. “OpenAI Gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[Dua+16] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: *arXiv preprint arXiv:1604.06778* (2016).

[Hee+17] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, et al. “Emergence of Locomotion Behaviours in Rich Environments”. In: *arXiv preprint arXiv:1707.02286* (2017).

[KL02] S. Kakade and J. Langford. “Approximately optimal approximate reinforcement learning”. In: *ICML*. Vol. 2. 2002, pp. 267–274.

[KB14] D. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[Mni+15] 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”. In: *Nature* 518.7540 (2015), pp. 529–533.

[Mni+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. “Asynchronous methods for deep reinforcement learning”. In: *arXiv preprint arXiv:1602.01783* (2016).

[Sch+15a] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015).

[Sch+15b] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. “Trust region policy optimization”. In: *CoRR, abs/1502.05477* (2015).

[SL06] I. Szita and A. Lörincz. “Learning Tetris using the noisy cross-entropy method”. In: *Neural computation* 18.12 (2006), pp. 2936–2941.

[TET12] E. Todorov, T. Erez, and Y. Tassa. “MuJoCo: A physics engine for model-based control”. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 5026–5033.

[Wan+16] Z. Wang, V. Babst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. “Sample Efficient Actor-Critic with Experience Replay”. In: *arXiv preprint arXiv:1611.01224* (2016).

[Wil92] R. J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

超参数

超参数	值
时间步长 (T)	2048
Adam 步长	3×10^{-4}
轮数	10
小批量大小	64
折扣因子 (γ)	0.99
GAE 参数 (λ)	0.95

表 3　用于 MuJoCo 100 万时间步基准测试的 PPO 超参数

超参数	值
时间步长（T）	512
Adam 步长	*
轮数	15
小批量大小	4096
折扣因子（γ）	0.99
GAE 参数（λ）	0.95
智能体数量	32（locomotion）, 128（flagrun）
动作分布的对数标准差	LinearAnneal(-0.7, -1.6)

表 4　用于 Roboschool 实验的 PPO 超参数。Adam 步长根据 KL 散度的目标值进行调整。

超参数	值
时间步长（T）	128
Adam 步长	$2.5 \times 10^{-4} \times \alpha$
轮数	3
小批量大小	32×8
折扣因子（γ）	0.99
GAE 参数（λ）	0.95
智能体数量	8
裁剪参数 ϵ	$0.1 \times \alpha$
VF 系数 c_1（式 (9)）	1
熵系数 c_2（式 (9)）	0.01

表 5　用于 Atari 实验的 PPO 超参数。α 在学习过程中从 1 线性衰减到 0。

更多雅达利游戏上的性能

在此，我们纳入了 PPO 与 A2C 在更多的 49 款雅达利游戏上的比较。图.6 展示了三个随机种子各自的学习曲线，而表 6 展示了平均性能。

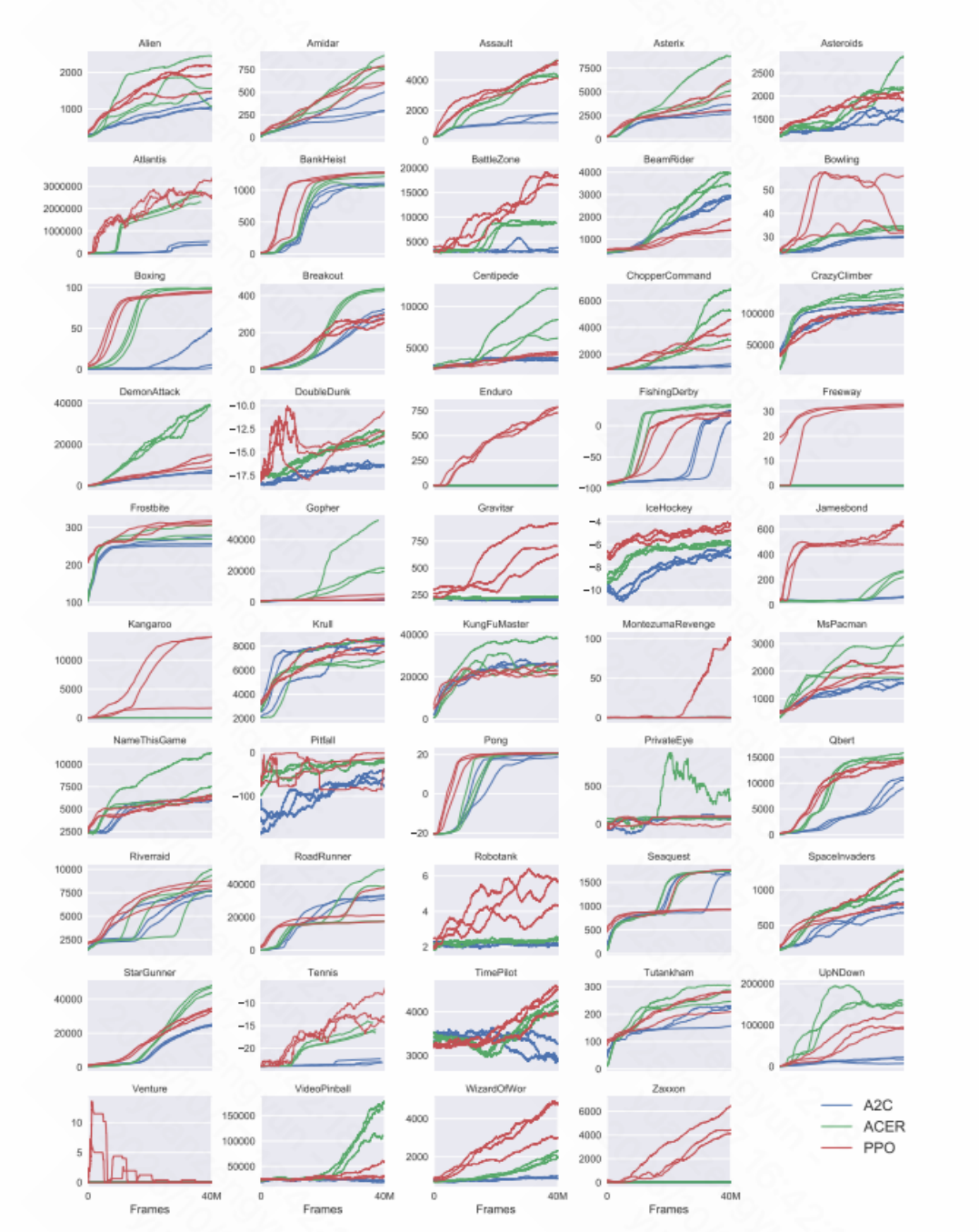


图 .6　在发表时，PPO 和 A2C 在 OpenAI Gym 中包含的所有 49 款雅达利游戏上的比较。

补充说明

关于损失函数的补充说明

平方误差项

在上面提到的损失函数中，$L_t^{VF}(\theta) = (V_\theta(s_t) - V_t^{\text{target}})^2$. 这里的 V^{target} 是用 Rollout Buffer 的样本数据计算得到的，是当前状态的目标回报

$$V_t^{\text{target}} = \sum_{i=t}^{T} \gamma^{i-t} r_i + \gamma^{T-t} V_\theta(s_T)$$

其中 T 是我们用旧策略采样的步数，即 Rollout Buffer 的大小.

因为 $\hat{A}_t^{GAE} = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) = -V(s_t) + V_t^{\text{target}}$ 再将这个等式参数化，这样我们就得到了一种计算方式，即

$$V_t^{\text{target}} = \hat{A}_t^{GAE} + V_{\theta_{\text{old}}}(s_t)$$

需要注意，这里的参数 θ 和损失函数中正在更新的 θ 并不一样，此处的 θ 是一个固定的旧参数，于是我们可以计算出平方误差项

$$L_t^{VF} = \left(V_\theta(s_t) - V_{\theta_{\text{old}}}(s_t) - \hat{A}_t^{GAE} \right)^2$$

熵奖励项

平均策略熵 $S[\pi_\theta]$ 作为正则化项添加在目标函数中，其中

单状态策略熵：

$$H(\pi_\theta(\cdot|s_t)) = \begin{cases} -\sum_{a \in \mathcal{A}} \pi_\theta(a|s_t) \cdot \log \pi_\theta(a|s_t) & \text{离散任务} \\ \dfrac{1}{2} \log(2\pi e \cdot \sigma_\theta^2(s_t)) & \text{连续任务} \end{cases}$$

平均策略熵：$S[\pi_\theta] = \mathbb{E}_{s_t \sim \pi_\theta} H(\pi_\theta(\cdot|s_t))$，即所有状态下单状态策略熵的数学期望.

损失函数可微性说明

在损失函数 $L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right]$ 中，由于 L_t^{CLIP} 的存在，其在 $r = 1 - \epsilon$ 或 $r = 1 + \epsilon$ 时很大程度上不可微，这在原理上为后续的优化带来了困难.

我们可以在工程上用两个观点解释这个问题：

1. 这两个点的测度为 0，这也就是说我们在实际优化时，计算出的 r 恰好落在特定点的概率极低，这也就保证了在大多数情况下，优化是可以正常进行的.
2. 相关的编程语言 (如 PyTorch) 的优化器在处理不可微点时，会选择其次梯度 (左导数或右导数)，这样就在工程上避免了零测不可微点的求导问题.

关于 Adam 算法的补充说明

Adam 优化器伪代码

算法 9 Adam 随机优化算法

输入: 步长 α
输入: 指数衰减率 $\beta_1, \beta_2 \in [0, 1)$（用于矩估计）
输入: 随机目标函数 $f(\theta)$（参数为 θ）
输入: 初始参数向量 θ_0

$m_0 \leftarrow 0$ ▷ 初始化一阶矩向量
$v_0 \leftarrow 0$ ▷ 初始化二阶原始矩向量
$t \leftarrow 0$ ▷ 初始化时间步
while θ_t 未收敛 **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ 计算时间步 t 的梯度
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ 更新有偏一阶矩估计
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ ▷ 更新有偏二阶原始矩估计（g_t^2 为元素-wise 平方）
  $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ 计算偏差校正后的一阶矩估计
  $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ 计算偏差校正后的二阶原始矩估计
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ ▷ 更新参数
end while
return θ_t ▷ 返回优化后的参数

注：1. 论文推荐默认参数设置：$\alpha = 0.001$，$\beta_1 = 0.9$，$\beta_2 = 0.999$，$\epsilon = 10^{-8}$；

2. 所有向量运算均为元素-wise 操作；

3. β_1^t 表示 β_1 的 t 次幂，$\sqrt{\widehat{v}_t}$ 为 $\widehat{v}_t$ 的元素-wise 平方根

Adam 算法的收敛性证明

见相关专栏 `https://zhuanlan.zhihu.com/p/345416641`

游戏名称	A2C	ACER	PPO
Alien	1141.7	1655.4	1850.3
Amidar	380.8	827.6	674.6
Assault	1562.9	4653.8	4971.9
Asterix	3176.3	6801.2	4532.5
Asteroids	1653.3	2389.3	2097.5
Atlantis	729265.3	1841376.0	2311815.0
BankHeist	1095.3	1177.5	1280.6
BattleZone	3080.0	8983.3	17366.7
BeamRider	3031.7	3863.3	1590.0
Bowling	30.1	33.3	40.1
Boxing	17.7	98.9	94.6
Breakout	303.0	456.4	274.8
Centipede	3496.5	8904.8	4386.4
ChopperCommand	1171.7	5287.7	3516.3
CrazyClimber	107770.0	132461.0	110202.0
DemonAttack	6639.1	38808.3	11378.4
DoubleDunk	-16.2	-13.2	-14.9
Enduro	0.0	0.0	758.3
FishingDerby	20.6	34.7	17.8
Freeway	0.0	0.0	32.5
Frostbite	261.8	285.6	314.2
Gopher	1500.9	37802.3	2932.9
Gravitar	194.0	225.3	737.2
IceHockey	-6.4	-5.9	-4.2
Jamesbond	52.3	261.8	560.7
Kangaroo	45.3	50.0	9928.7
Krull	8367.4	7268.4	7942.3
KungFuMaster	24900.3	27599.3	23310.3
MontezumaRevenge	0.0	0.3	42.0
MsPacman	1626.9	2718.5	2096.5
NameThisGame	5961.2	8488.0	6254.9
Pitfall	-55.0	-16.9	-32.9
Pong	19.7	20.7	20.7
PrivateEye	91.3	182.0	69.5
Qbert	10065.7	15316.6	14293.3
Riverraid	7653.5	9125.1	8393.6
RoadRunner	32810.0	35466.0	25076.0
Robotank	2.2	2.5	5.5
Seaquest	1714.3	1739.5	1204.5
SpaceInvaders	744.5	1213.9	942.5
StarGunner	26204.0	49817.7	32689.0
Tennis	-22.2	-17.6	-14.8
TimePilot	2898.0	4175.7	4342.0
Tutankham	206.8	280.8	254.4
UpNDown	17369.8	145051.4	95445.0
Venture	0.0	0.0	0.0
VideoPinball	19735.9	156225.6	37389.0
WizardOfWor	859.0	2308.3	4185.3
Zaxxon	16.3	29.0	5008.7

表 6　在 4000 万游戏帧（1000 万时间步）后，PPO 和 A2C 在雅达利游戏上的平均最终得分（最后 100 个回合）。

丰富环境中运动行为的涌现[1,2]

摘要

强化学习范式在理论上允许智能体从简单的奖励信号中自主学习复杂行为。但在实际应用中，精心手工设计奖励函数以促成特定解决方案，或是从演示数据中推导奖励函数，是较为常见的做法。本文探索：一个丰富的环境如何推动复杂行为的学习。具体而言，我们在多样化的环境场景中训练智能体，发现这一方式能促使稳健行为的涌现，且智能体可在一系列任务中泛化此类行为。我们以移动行为为例验证这一原则——这类行为因对奖励函数的选择高度敏感而闻名。我们基于“向前推进”这一简单奖励函数，在包含障碍与复杂地形的多样环境中训练多个带扭矩控制的身体模型。通过一种新的可扩展强化学习策略梯度算法，智能体自主习得奔跑、跳跃、蹲伏与转向等行为，且无需显式的奖励引导。已学习行为的亮点可视化可通过此视频查看。

1 引言

强化学习已取得显著进展，在雅达利游戏[1]、三维导航任务[2,3]与棋类游戏[4]等领域达到了高性能水平。这些任务的共同特点是存在定义明确的奖励函数（例如游戏得分），可通过优化生成期望行为。然而，在许多其他任务中，“正确”的奖励函数并不清晰，随意选择的奖励函数可能产生与设计者预期不符的意外结果。这一问题在连续控制任务中尤为突出，移动行为便是典型代表：目前的常规做法是精心手工设计奖励函数，或是从演示数据中推导奖励函数。

然而，这些移动行为的成功演示案例具有脆弱性：若奖励函数发生轻微调整，可能导致意外结果；此外，对于更复杂的行为，恰当奖励函数的设计本身往往并不直观。更根本地说，这回避了强化学习的核心挑战之一：智能体如何仅从有限的奖励信号中自主学习，以实现丰富且有效的行为。本文将回归这一挑战。

我们的前提是：丰富且稳健的行为可从简单奖励函数中涌现。首先，若环境本身包含不同难度级别的各类挑战，可能会塑造学习过程，并引导智能体找到在更受限场景中难以发现的解决方案。其次，寻找奖励函数与其他实验细节的做法，可能会导致智能体过度拟合特定场景下恰好有效的独特解决方案，但这类方案在

[1]原文：Emergence of Locomotion Behaviours in Rich Environments, Heess et al, 2017. Algorithm: PPO-Penalty.

[2]译者：贾顺。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

智能体面临更广泛场景时缺乏稳健性。为智能体提供多样化的挑战，会拉大不同解决方案间的性能差距，进而更有利于学习出能在各类场景下稳健泛化的方案。

我们聚焦于一系列全新的移动任务，这些任务显著超越了现有直接通过强化学习训练智能体的技术水平。任务包含针对不同身体模型（四足机器人（Quadruped）、平面步行器（Planar Walker）、人形机器人（Humanoid）[5,6]）的多种障碍场景，场景通过程序化生成，因此每个回合都会呈现该任务的不同实例。

我们的环境包含各类障碍，难度等级各不相同（例如坡度、路面不平度、缺口间距）。难度的变化为智能体构建了“隐性课程”：随着智能体能力提升，它能够应对日益复杂的挑战，最终涌现出原本看似需要精心设计奖励函数或其他指导方法才能实现的复杂移动技能。我们还发现，通过显式构建“难度逐步提升”的地形（让智能体先面对简单障碍，掌握后再挑战复杂障碍），可以加快学习速度。

为了在这类丰富且具有挑战性的领域中高效学习，需要可靠且可扩展的强化学习算法。我们借鉴了近期深度强化学习领域的鲁棒策略梯度方法：首先，采用信任域策略优化（TRPO）与近端策略优化（PPO[7,8]）等方法——这类方法通过将参数更新约束在信任域内保证稳定性；其次，参考广泛应用的异步优势演员-评论家（A3C）算法[2] 及相关方法[3]，将计算过程分布到多个并行的智能体实例与环境中。我们的分布式 PPO 实现相较于 TRPO，在智能体数量方面的耗时大幅减少，且稳健性基本一致；同时，在工作节点数量相同的情况下，相较于我们现有针对连续动作的 A3C 实现，性能也有所提升。

本文结构如下：第 2 节介绍分布式 PPO（DPPO）算法，该算法支撑后续实验，我们将通过实证验证其有效性；第 3 节介绍主要实验设置：多样化的复杂地形与障碍；第 4 节提供证据表明，有效的移动行为可直接从简单奖励中涌现；此外，我们还将展示，带有“难度课程”的地形能大幅加快学习进度，且在更多样环境中训练的智能体具有更强的稳健性。

2 基于分布式 PPO 的大规模强化学习

我们的研究聚焦于“状态与动作空间连续”的丰富仿真环境中的强化学习。我们需要的算法需满足两个条件：一是能在任务的广泛变化中保持稳健性，二是能有效扩展至具有挑战性的领域。我们将依次解决这些问题。

基于近端策略优化的鲁棒策略梯度　基于大规模高吞吐量优化方法的深度强化学习算法，已在离散、低维动作空间中取得了最先进的成果，例如在雅达利游戏[9]与三维导航任务[2,3] 中的应用。相比之下，现有针对连续动作空间的研究（如[6,7,10–13]）虽成果显著，但多聚焦于相对较小的问题；而大规模分布式优化的应用较为有限，

相关算法的发展也不够成熟（但可参考[14–16]）。本文提出一种鲁棒的策略梯度算法，适用于高维连续控制问题，且可通过分布式计算扩展至更大的领域。

策略梯度算法[17]为连续控制提供了一种具有吸引力的范式。这类算法通过直接最大化期望奖励和 $J(\theta) = \mathbb{E}_{\rho_\theta(\tau)}\left[\sum_t \gamma^{t-1} r(s_t, a_t)\right]$ 来优化参数 θ（$\pi_\theta(a|s)$ 为随机策略）。其中，期望是针对系统动力学生成的轨迹 $\tau = (s_0, a_0, s_1, a_1, \dots)$ 的分布而言的；轨迹分布 $\rho_\theta(\tau)$ 由初始状态分布 $p(s_0)$、策略 π_θ 与系统动力学 $p(s_{t+1}|s_t, a_t)$ 共同决定，即 $\rho_\theta(\tau) = p(s_0)\pi(a_0|s_0)p(s_1|s_0, a_0)\dots$。

目标函数关于 θ 的梯度为 $\nabla_\theta J = \mathbb{E}_\theta\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - b_t)\right]$，其中 $R_t = \sum_{t'=t} \gamma^{t'-t} r(s_{t'}, a_{t'})$，$b_t$ 是不依赖于 a_t 及未来状态与动作的基准函数。基准函数通常选取为 $b_t = V^\theta(s_t) = \mathbb{E}_\theta[R_t|s_t]$。在实际应用中，期望回报通常通过采样轨迹近似，而 V^θ 由带参数 ϕ 的学习近似函数 $V_\phi(s)$ 替代。

策略梯度估计的方差可能较高（例如[18]），且算法对超参数的设置较为敏感。目前已有多种方法被提出以提升策略梯度算法的稳健性，其中一种有效措施是引入“信任域约束”，限制策略更新的幅度[7,14,19]。

一个应用这一思想的主流算法是信任域策略优化（TRPO）[7]。在每个迭代步骤中，给定当前参数 θ_{old}，TRPO 会采集一个（相对较大的）数据批次，然后优化代理损失函数：

$$J_{\text{TRPO}}(\theta) = \mathbb{E}_{\rho_{\theta_{\text{old}}}(\tau)}\left[\sum_t \gamma^{t-1} \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A^{\theta_{\text{old}}}(a_t, s_t)\right]$$

该优化过程需满足策略变化幅度的约束（以 KL 散度衡量）：$\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta] < \delta$。其中，$A^{\theta_{\text{old}}}$ 为优势函数，定义为 $A^{\theta_{\text{old}}}(s_t, a_t) = \mathbb{E}_\theta[R_t|s_t, a_t] - V^{\theta_{\text{old}}}(s_t)$。

近端策略优化（PPO）算法[8]可视为 TRPO 的近似版本，它仅依赖一阶梯度，因此更便于与循环神经网络（RNN）结合使用，也更适配大规模分布式场景。PPO 通过正则化项实现信任域约束，该正则化项的系数会根据约束是否被违反进行自适应调整（类似思想也被用于[13]，但未采用自适应系数）。算法框 1 展示了 PPO 的核心伪代码。

在算法10中，超参数 $\text{KL}_{\text{target}}$ 表示每次迭代中策略期望的变化量。缩放项 $\alpha > 1$ 用于控制 KL 正则化系数的调整：当策略的实际变化量显著低于或高于目标 KL（即处于区间 $[\beta_{\text{low}}\text{KL}_{\text{target}}, \beta_{\text{high}}\text{KL}_{\text{target}}]$ 之外）时，会触发这一调整。

基于分布式 PPO 的可扩展强化学习　为了在丰富的仿真环境中实现良好性能，我们实现了 PPO 算法的分布式版本（DPPO）。数据采集与梯度计算的任务被分配给多个工作器。我们同时实验了同步更新与异步更新策略，实践发现：对梯度进行平均后再执行同步更新，能取得更优的结果。

算法 10 近端策略优化（改编自[8]）

for $i \in \{1, \ldots, N\}$ **do**
  运行策略 π_θ 共 T 个时间步，收集 $\{s_t, a_t, r_t\}$
  估计优势函数 $A_t = \sum_{t' \geq t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
  $\pi_{\text{old}} \leftarrow \pi_\theta$
  for $j \in \{1, \ldots, M\}$ **do**
    $J_{\text{PPO}}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \hat{A}_t - \lambda \text{KL}[\pi_{\text{old}}||\pi_\theta]$
    基于梯度方法，对 $J_{\text{PPO}}(\theta)$ 更新 θ
  end for
  for $j \in \{1, \ldots, B\}$ **do**
    $L_{\text{BL}}(\phi) = -\sum_{t=1}^{T} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - V_\phi(s_t) \right)^2$
    基于梯度方法，对 $L_{\text{BL}}(\phi)$ 更新 ϕ
  end for
  if $\text{KL}[\pi_{\text{old}}||\pi_\theta] > \beta_{\text{high}} \text{KL}_{\text{target}}$ **then**
    $\lambda \leftarrow \alpha\lambda$
  else if $\text{KL}[\pi_{\text{old}}||\pi_\theta] < \beta_{\text{low}} \text{KL}_{\text{target}}$ **then**
    $\lambda \leftarrow \lambda/\alpha$
  end if
end for

原始 PPO 算法通过奖励的完整求和来估计优势函数。为了在支持变长回合的同时，适配循环神经网络（RNN）的批量更新，我们采用了与文献[2] 相似的策略：使用时间截断反向传播，窗口长度设为 K。这使得我们自然（尽管非必需）地采用 K-步回报窗口来估计优势函数，即对奖励求和后，再加上 K 步后价值函数的 γ^{K-1} 倍，并减去当前状态的价值函数：

$$\hat{A}_t = \sum_{i=1}^{K} \gamma^{i-1} r_{t+i} + \gamma^{K-1} V_\phi(s_{t+K}) - V_\phi(s_t).$$

John Schulman 公开的 PPO 实现[20] 对核心算法做了若干改进，包括输入与奖励的归一化、以及损失函数中增加了惩罚信任域约束大幅违反的项。我们在分布式设置中采用了类似的增强策略，但跨工作器共享同步状态需要额外注意。我们的分布式 PPO（DPPO）实现基于 TensorFlow 框架：参数存储在参数服务器中，每个工作器在完成每一步梯度计算后，会同步其参数。伪代码及更多细节见补充材料。

2.1 分布式 PPO 的评估

我们将 DPPO 与若干基准算法进行对比。这些实验的核心目标是验证：该算法仅需少量参数调优即可实现鲁棒的策略优化，且能有效扩展到目标任务。因此，我们在与研究相关的选定基准任务上，将其与两种算法备选方案进行对比：TRPO 和连续动作空间的 A3C。对比细节详见补充材料。

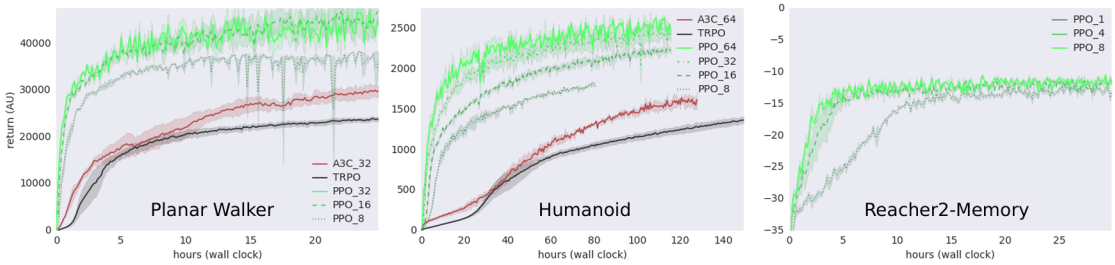


图 .7　DPPO 在平面步行器（左）、人形机器人（中）与记忆抓取器（右）任务上的基准性能。在所有任务中，DPPO 的性能与 TRPO 相当，且其性能随工作器数量增加而良好扩展。记忆抓取器任务表明，该方法可与循环神经网络配合使用。

基准任务　我们选取了三个连续控制任务用于算法基准测试。所有环境均基于 MuJoCo 物理引擎实现[21]。其中两个是无障碍环境下的移动任务，第三个是需要记忆能力的平面目标抓取任务：**平面步行器**：一种简单的双足步行器，具有 9 个自由度（DoF）和 6 个扭矩驱动关节。它的主要奖励与前进速度成正比；额外项用于惩罚控制违规（如关节角度超限），以及步行器摔倒时的提前回合终止。**人形机器人**：具有 28 个自由度和 21 个驱动关节。其主要奖励同样与 x 轴方向的速度成正比，同时包含一个鼓励不摔倒的常数奖励；每一步奖励会在摔倒时清零，且回合立即终止。**记忆抓取器**：平面内的随机目标抓取任务，采用具有 2 个自由度的简单机械臂。目标位置仅在每个回合的前 10 步提供，之后目标消失且机械臂不能移动。此时需依赖循环神经网络（RNN）的记忆能力，才能让机械臂朝向正确目标移动。该任务的奖励是末端执行器与目标位置的距离（需最小化），用于测试策略优化循环神经网络的能力。

结果　图 1 所示结果表明：DPPO 的性能与 TRPO 相当，且其性能随工作器数量增加而良好扩展（可显著缩短实际运行时间）。由于它完全基于梯度实现，因此可直接与循环神经网络配合使用（记忆抓取器任务已验证这一点）。在使用相同数量工作器的情况下，DPPO 的实际运行速度也快于我们实现的 A3C。

3 方法：环境与模型

我们的目标是研究：当智能体从包含不同难度的多样化挑战中学习时，仅通过简单奖励能否涌现出复杂的移动技能。在简单基准任务上验证了可扩展的 DPPO 算法后，接下来我们将描述用于展示更复杂行为涌现的实验设置。

3.1 训练环境

为了让智能体接触多样化的移动挑战，我们使用了一个与平台游戏大致类似的物理仿真环境（同样基于 MuJoCo 实现[21]）。我们通过程序生成大量不同地形和

障碍物；每个回合都会生成地形与障碍物的不同实例。

身体模型 我们考虑三种不同的扭矩控制身体模型，其复杂度大致递增：**平面步行器**：受限于平面的简单步行体，具有 9 个自由度（DoF）和 6 个驱动关节。**四足机器人**：简单的三维四足体，具有 12 个自由度和 8 个驱动关节。**人形机器人**：三维人形体，具有 21 个驱动维度和 28 个自由度。

这些身体模型的运行效果分别见图 4、5、7。需注意：平面步行器和人形机器人已用于前文所述的基准任务，但这些基准任务仅包含开放平面中的简单移动。

奖励 我们将所有任务的奖励函数保持简单，且在不同地形中保持一致。奖励包含两部分：一是与 x 轴方向速度成正比的主成分，用于鼓励智能体沿轨道前进；二是用于正则化扭矩的小惩罚项。对于步行器，奖励还包含与第 2 节中相同的位姿约束项。对于四足机器人和人形机器人，我们会惩罚其偏离轨道中心的行为；人形机器人每一步不摔倒还会获得额外奖励。详细信息见补充材料。我们注意到：不同身体模型的奖励函数差异，是我们采用前文提出的奖励函数（例如[12,18]）的结果，而非精心调参的产物；尽管不同身体模型的奖励函数略有差异，但对于同一身体模型，我们不会通过修改奖励函数来诱导不同行为。

地形与障碍物 我们所有的轨道均通过程序生成；每个回合会基于预定义的统计特征生成新轨道。我们考虑多种不同的地形与障碍物类型：(a) 障碍栏：高度与宽度可变的栏状障碍，步行体需跳跃或攀爬通过；(b) 间隙：地面上的空隙，需跳跃通过；(c) 可变地形：包含坡道、间隙、山丘等特征的地形；(d) 绕桩墙：形成障碍的墙体，需绕行通过；(e) 悬浮平台：地面上方的平台，可跳跃至平台上或蹲伏通过。轨道由上述地形类型的随机实例序列组成，实例参数处于用户指定的范围内。

我们在不同类型的轨道上训练：单一类型轨道（如仅间隙、仅障碍栏等）；混合单一类型轨道（如每个回合选择不同的地形类型）；混合地形（单一轨道包含多种地形类型）。我们还考虑静态轨道（障碍统计特征在轨道全程基本固定），以及“课程式”轨道（地形难度沿轨道全程逐步提升）。图 3 展示了部分不同类型的轨道。

观测信息 智能体接收两类观测信息[22]：(1) 以自身为中心的“本体感受”特征：包含关节角度与角速度；对于四足机器人和人形机器人，这些特征还包含躯干处速度计、加速度计和陀螺仪的读数（提供以自身为中心的速度、位置与加速度信息），以及足部和腿部的触觉传感器数据。人形机器人的关节处还装有扭矩传感器。(2) “外感受”特征：包含与任务相关的信息，包括相对于轨道中心的位置，以及前方地形的轮廓；地形信息还包含采样点的高度测量数组，这些采样点随身体沿 x 轴平移，采样密度随与身体的距离增加而降低。平面步行器被限制在 xz 平

面内（即无法左右移动），因此其本体感受特征会更简单。详细信息见补充材料。

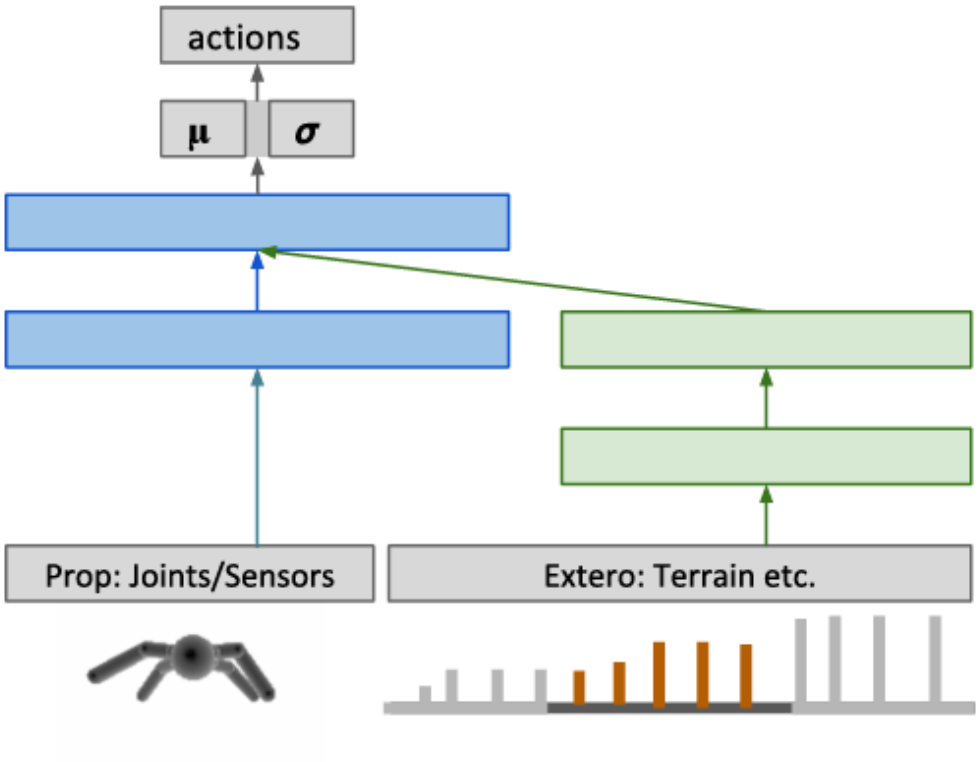


图 .8　网络架构示意图。我们采用了与文献[22]相似的架构，包含两个组件：一个组件（蓝色）处理受控身体的以自身为中心的局部信息（本体感受信息）；另一个调节组件（绿色）处理环境及任务相关的“外感受”信息（如地形形状）。

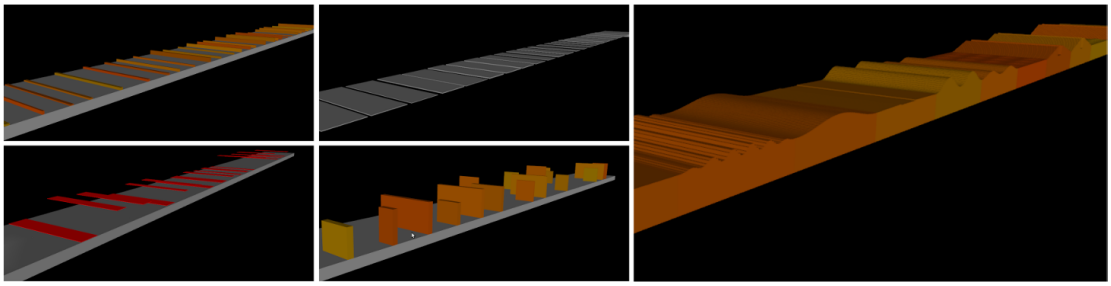


图 .9　实验中所用地形类型的示例。从左到右、从上到下依次为：障碍栏、悬浮平台、间隙、绕桩墙、可变地形。

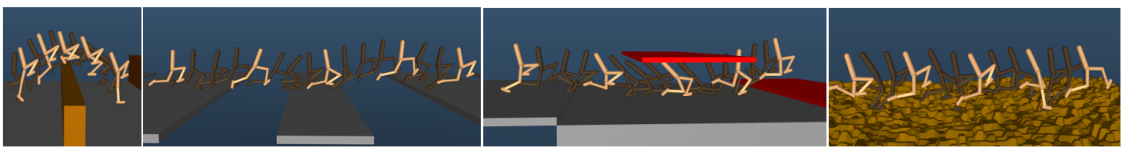


图 .10　步行器技能：平面步行器策略穿越碎石地形、跳跃障碍栏、跨越间隙、蹲伏通过悬浮平台下方的代表性时序图像。

3.2 策略参数化

与文献[22]相似，我们旨在将基础移动技能与地形感知、导航这两类功能分离。我们将策略构建为两个子网络：一个仅接收本体感受信息，另一个仅接收外感受信息。如前文所述，本体感受信息是独立于任务的、身体局部的信息，而外感受信息包含前方地形的表示。我们将该架构与简单的全连接神经网络进行了对比，发现它大幅提升了学习速度。图 2 展示了架构示意图。

4 结果

我们将分布式 PPO 算法应用于多种身体模型、地形与障碍物。我们的目标是验证：当智能体在丰富环境中训练时，简单的奖励函数能否引导复杂移动技能的涌现。我们还进一步探究地形结构是否会影响学习效果及最终行为的鲁棒性。

平面步行器 我们分别在障碍栏、间隙、悬浮平台、可变地形上训练步行器，也在包含所有特征的混合轨道上训练，还在混合地形（即步行器在不同回合中处于不同地形）上训练。它掌握了稳健的步态：学会了跳跃障碍栏与间隙、蹲伏通过悬浮平台下方。所有这些行为都是自发出现的，未通过特殊奖励来诱导每种独立行为。图 4 展示了平面步行器穿越碎石地形、跳跃障碍栏、跨越间隙、蹲伏通过悬浮平台的动作序列。这些行为在不同随机种子下均能复现，补充视频中也展示了其稳健的移动能力。训练结束时，平面步行器能跃过接近自身身高的障碍栏。

四足机器人 四足机器人的灵活性通常低于步行器，但它为控制问题增加了第三个维度。我们考虑了三种地形类型：可变地形、绕桩墙、间隙，以及一种包含可攀爬或跳跃障碍物的障碍栏地形变体。

四足机器人也能较可靠地导航大多数障碍物，不同随机种子下的表现差异很小。它发现跳跃（向上或向前，部分情况下精度惊人）是克服障碍栏与间隙的有效策略，还学会了前进、左转、右转——尽管它仅能获得前进奖励。在障碍栏地形变体中，它学会了区分可攀爬/需攀爬的障碍物与需绕行的障碍物。可变地形看似简单，实际难度却出人意料，这是因为四足机器人的身体形态并不适配（即四足机器人的腿部相对于地形变化而言较短）。尽管如此，它仍学会了相对稳健的穿越策略。图 5 展示了部分代表性动作序列；更多示例见补充视频。

分析 我们探究地形特性是否会影响学习过程。例如，很容易想到：仅在极高的障碍栏地形上训练是无效的。在我们的设置中，训练要取得成功，需要步行器偶尔能“偶然解决”障碍物——即轨道中偶尔出现极高障碍栏的概率是存在的。我们通过在两种不同的障碍栏地形上训练平面步行器来验证这一点：第一种地形的障碍栏高度是固定的，高障碍栏与低障碍栏随机交错；第二种地形的障碍栏难

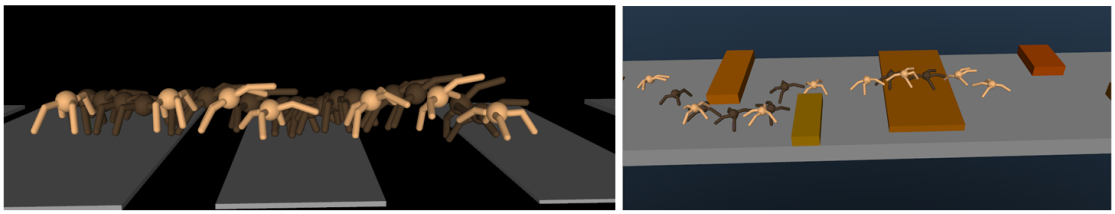


图 .11　四足机器人策略跨越间隙（左）与导航障碍物（右）的代表性时序图像。

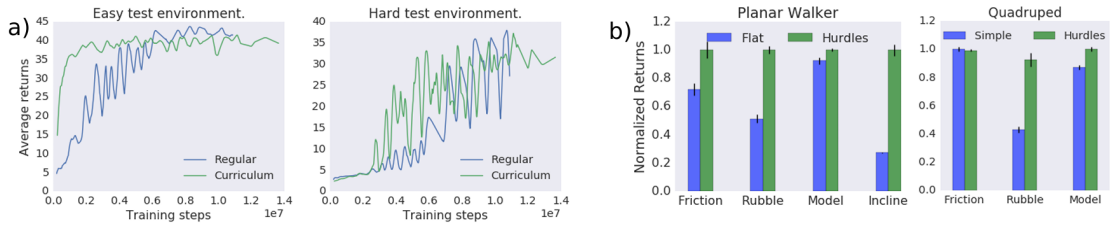


图 .12　a) 课程式训练：对在不同统计特征的障碍栏轨道上训练的策略进行评估：“常规”轨道包含随机交错的高、低障碍栏（蓝色）；“课程式”轨道沿轨道长度逐渐增加障碍栏高度（绿色）。训练期间，我们在低障碍栏的“简单”轨道（左）和高障碍栏的“困难”轨道（右）上评估这两种策略。在课程式轨道上训练的策略，其性能提升速度更快。b) 平面步行器策略（左）与四足机器人策略（右）的稳健性：我们评估在障碍栏地形上训练（绿色）相比在平坦地形上训练（蓝色），策略稳健性的提升情况。我们通过地形表面（碎石）、身体执行器强度、地面平面倾斜等未观测变化的场景来评估策略。在部分场景中，在障碍栏地形上训练的策略具有显著优势。所有图均展示了各地形设置下归一化后的平均回报。

度会沿轨道长度逐渐增加，障碍栏的最小与最大高度随之上升。我们通过在测试地形（一个是含浅障碍栏的简单地形，另一个是含高障碍栏的困难地形）上评估策略，来衡量学习进度。图 6a 展示了代表性平面步行器策略的结果：在难度逐渐增加的地形上训练的策略，其提升速度快于在固定地形上训练的策略。

我们进一步研究：与在平面上前进的常规任务相比，在多样地形上训练是否能产生更稳健的步态。为此，我们在平坦轨道和（更具挑战性的）障碍栏轨道上分别训练平面步行器与四足机器人的策略，然后评估各实验中代表性策略的稳健性，测试场景包括：(a) 未观测到的地面摩擦变化，(b) 未观测到的碎石路面，(c) 身体模型的随机扰动，(d) 未观测到的地面倾斜/下降。图 6b 的结果显示：在障碍栏地形上训练，会提升策略对地形中其他未观测变化的稳健性。

人形机器人　我们的最后一组实验针对 28 自由度的人形机器人——其复杂度远高于平面步行器与四足机器人。所用地形类型与其他身体模型的地形定性相似，包括间隙、障碍栏、可变地形及绕桩墙。我们也在上述地形的混合轨道上训练了智能体。

与之前的实验一致，我们采用简单的奖励函数，其主成分与 x 轴方向的速度成正比（见前文）。我们测试了两种备选终止条件：(a) 当头部与地面的最小距离小于 0.9m 时，回合终止；(b) 当头部与足部的最小距离小于 1m 时，回合终止。

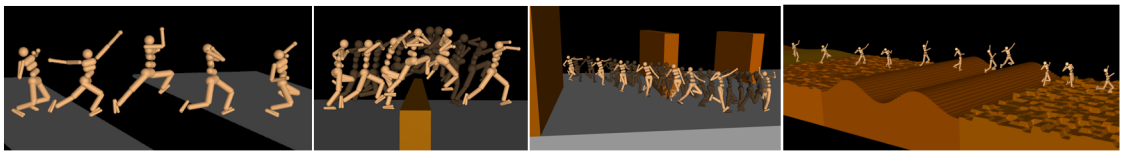


图 .13　人形机器人导航不同地形的时序图像序列

总体而言，人形机器人的学习问题难度显著更高——主要是因为其自由度数量较多，容易利用任务描述中的冗余信息，或陷入局部最优，从而产生有趣但视觉效果不佳的步态。学习结果往往对初始化、具体算法、探索策略、奖励函数、终止条件及权重初始化较为敏感。

我们得到的人形机器人结果，其多样性远高于另外两种身体模型：在超参数设置相同的情况下，不同随机种子对应的结果差异显著。部分行为差异与学习速度、渐近性能的不同相关（暗示存在行为最优解），另一部分差异则对应备选解决方案（暗示存在多解性）。

尽管如此，我们仍在每种地形上得到了若干性能良好的智能体——无论从性能还是视觉上美观的步态来看均是如此。图 7 展示了在间隙、障碍栏、绕桩墙、可变地形上训练的智能体示例。与之前的实验一致，地形多样性及内在课程引导智能体掌握了稳健的技能：能够跨越障碍物、跳跃间隙、导航绕桩墙。我们在补充视频中突出展示了每种地形的多种解决方案（包括视觉效果较差的方案）。为测试所学行为的稳健性，我们构建了两个测试轨道：(a) 统计特征与训练地形差异较大的轨道；(b) 以随机力施加在人形机器人躯干上的未观测扰动。相关内容也在视频中呈现。从定性角度看，这些测试挑战下的稳健性水平较高（见补充视频）。

5 相关工作

基于物理的角色动画是一个长期活跃的研究领域，已产出大量成果，为仿真角色赋予了移动及其他动作技能（综述见[23]）。例如，文献[24] 展示了在参数化地形上操控障碍物的复杂技能序列；文献[25–27] 展示了自适应移动或熟练动作如何从优化问题中涌现。尽管方法多样，但本质上都依赖于问题领域的大量先验知识，以及动作捕捉数据等演示信息。

基于强化学习的端到端基础移动行为已被实现，例如文献[6,7,12,13] 的成果，或文献[10] 的引导策略搜索。文献[22] 考虑了更高级任务下的移动问题。文献[28] 展示了基于强化学习的地形自适应移动，但仍对解决方案施加了大量结构约束。近期，文献[29] 在 3D 人形身体的学习移动控制器上取得了令人印象深刻的结果，但这些方法依赖于特定结构和人类动作捕捉数据来引导技能，而这些技能仅适用于平坦地形导航。

课程学习的思想在机器学习文献中由来已久（例如[30]），并已被用于学习动作技能，例如文献[31]的工作。本文结合并发展了这些研究方向的元素，但在特定方向上实现了独特的突破——仅对策略和行为施加有限的结构约束，通过简单的强化学习奖励与课程训练，在具有挑战性的环境中生成自适应移动行为。

6 讨论

我们探究了如下问题：在丰富环境中训练智能体，能否以及在多大程度上促使未被奖励函数直接激励的行为涌现。这与控制领域的常见设置不同——后者会精心调整奖励函数以实现特定解决方案。而我们刻意采用简单通用的奖励函数，同时在广泛的环境条件下训练智能体。

实验表明，在多样地形上训练确实能让智能体掌握非平凡的移动技能，如跳跃、蹲伏、转向等，而这些技能的设计并不容易。我们并非宣称环境变化足以替代合理的奖励函数设计，但我们认为：在比当前常见任务更广泛的任务谱上训练智能体，有望提升所学行为的质量与稳健性，同时降低学习难度。从这个意义上说，选择看似更复杂的环境，实际上可能让学习过程更简单。

致谢

感谢 DeepMind 的 Joseph Modayil 及其他多位同事，他们为本文提供了有益的讨论与修改意见。

参考文献

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[2] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In ICML, 2016.

[3] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. arXiv preprint arXiv:1611.05397, 2016.

[4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.

[5] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 4906–4913. IEEE, 2012.

[6] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438, 2015.

[7] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In ICML, pages 1889–1897, 2015.

[8] Pieter Abbeel and John Schulman. Deep reinforcement learning through policy optimization. Tutorial at Neural Information Processing Systems, 2016. URL `https://nips.cc/conferences/2016/Schedule?showEvent=6198`.

[9] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. arXiv preprint arXiv:1507.02296, 2015.

[10] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In NIPS, 2014.

[11] S Levine, C Finn, T Darrell, and P Abbeel. End-to-end training of deep visuomotor policies. arXiv preprint arXiv:1504.00702, 2015.

[12] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[13] Nicolas Heess, Gregory Wayne, David Silver, Timothy P Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In NIPS, 2015.

[14] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. CoRR, abs/1611.01224, 2016.

[15] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. arXiv preprint arXiv:1610.00633, 2016.

[16] Ivaylo Popov, Nicolas Heess, Timothy P. Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin A. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. CoRR, abs/1704.03707, 2017.

[17] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229–256, 1992.

[18] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. CoRR, abs/1604.06778, 2016.

[19] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010), 2010.

[20] PPO. `https://github.com/joschu/modular_rl`, 2016.

[21] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 5026–5033. IEEE, 2012.

[22] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning transfer of modulated locomotor controllers. arXiv preprint `arXiv:1610.05182`, 2016.

[23] Thomas Geijtenbeek and Nicolas Pronost. Interactive character animation using simulated physics: A state-of-the-art review. In Computer Graphics Forum, volume 31, pages 2492–2515. Wiley Online Library, 2012.

[24] Libin Liu, KangKang Yin, Michiel van de Panne, and Baining Guo. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. ACM Transactions on Graphics (TOG), 31(6):154, 2012.

[25] Jia-chi Wu and Zoran Popović. Terrain-adaptive bipedal locomotion control. ACM Transactions on Graphics, 29(4):72:1–72:10, Jul. 2010.

[26] Igor Mordatch, Martin De Lasa, and Aaron Hertzmann. Robust physics-based locomotion using low-dimensional planning. ACM Transactions on Graphics (TOG), 29(4):71, 2010.

[27] Igor Mordatch, Emanuel Todorov, and Zoran Popovic. Discovery of complex behaviors through contact-invariant optimization. ACM Trans. Graph., 31(4):43:1–43:8, 2012.

[28] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. ACM Transactions on Graphics (Proc. SIGGRAPH 2016), 35(4), 2016.

[29] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Transactions on Graphics (Proc. SIGGRAPH 2017), 36(4), 2017.

[30] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In International Conference on Machine Learning, ICML, 2009.

[31] Andrej Karpathy and Michiel Van De Panne. Curriculum learning for motor skills. In Canadian Conference on Artificial Intelligence, pages 325–330. Springer, 2012.

A 分布式 PPO

A.1 算法细节

分布式 PPO 算法的伪代码见算法框 2 和 3。W 为工作器数量；D 是梯度需就绪的工作器数量阈值；M、B 是策略和基线参数的子迭代次数；T 是每个工作器在计算参数更新前收集的数据点数量；K 是计算 K-步回报与时间截断反向传播（适用于 RNN）的时间步数。

算法 11 分布式近端策略优化（chief）

for $i \in \{1, \ldots, N\}$ **do**
  for $j \in \{1, \ldots, M\}$ **do**
    等待，直到至少有 $W - D$ 个关于 θ 的梯度就绪
    平均梯度并更新全局 θ
  end for
  for $j \in \{1, \ldots, B\}$ **do**
    等待，直到至少有 $W - D$ 个关于 ϕ 的梯度就绪
    平均梯度并更新全局 ϕ
  end for
end for

归一化处理 遵循文献[20]，我们执行如下归一化步骤：1. 对观测（或状态 s_t）进行归一化：利用整个实验过程中聚合的统计量，减去均值后除以标准差。2. 对奖励进行缩放：基于奖励的滑动标准差估计（同样利用整个实验过程中聚合的统计量）。3. 对优势函数进行批次内归一化。

工作器间的算法参数共享 在分布式设置中，我们发现跨工作器共享数据归一化的相关统计量十分重要。归一化在数据采集阶段生效，统计量会在每一步环境交互后在本地更新；当一次迭代的数据采集完成后，本地统计量的变化会同步到全局统计量（伪代码中未展示）。时变正则化参数 λ 也在工作器间共享，但其更新基于每个工作器本地计算的平均 KL 散度统计量，并由每个工作器单独应用调整后的 $\tilde{\alpha} = 1 + (\alpha - 1)/K$。

额外的信任域约束 我们还引入了额外的惩罚项：当 KL 散度超过期望变化量一定幅度时（本实验中阈值为 $2\mathrm{KL}_{\text{目标}}$），该惩罚项会生效。在分布式实现中，这一判定条件会在每个工作器上单独测试并应用。当 KL 散度的变化过大时，通过提前终止可进一步提升算法稳定性。

A.2 算法对比

TRPO 已被证实是一种稳健的算法：它能学习高性能策略，且仅需少量参数调优。因此，我们的核心关注点是 DPPO 能否取得与 TRPO 相当的结果；其次，我

算法 12 分布式近端策略优化（worker）

for $i \in \{1, \ldots, N\}$ **do**
  for $w \in \{1, \ldots, T/K\}$ **do**
    运行策略 π_θ 共 K 个时间步，收集 $\{s_t, a_t, r_t\}$（$t \in \{(i-1)K, \ldots, iK-1\}$）
    估计回报 $R_t = \sum_{t'=(i-1)K}^{iK-1} \gamma^{t'-(i-1)K} r_{t'} + \gamma^K V_\phi(s_{iK})$
    估计优势函数 $A_t = R_t - V_\phi(s_t)$
    存储部分轨迹信息
  end for
  $\pi_{\text{旧}} \leftarrow \pi_\theta$
  for $m \in \{1, \ldots, M\}$ **do**
    $J_{\text{近端策略优化}}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{旧}}(a_t|s_t)} \hat{A}_t - \lambda \text{KL}[\pi_{\text{旧}}||\pi_\theta] - \max(0, \text{KL}[\pi_{\text{旧}}||\pi_\theta] - 2\text{KL}_{\text{目标}})^2$
    if $\text{KL}[\pi_{\text{旧}}||\pi_\theta] > 4\text{KL}_{\text{目标}}$ **then**
      中断并继续下一个外层迭代 $i+1$
    end if
    计算 $\nabla_\theta J_{\text{近端策略优化}}$
    将关于 θ 的梯度发送至主节点
    等待梯度被接受或丢弃；更新参数
  end for
  for $b \in \{1, \ldots, B\}$ **do**
    $L_{\text{基线}}(\phi) = -\sum_{t=1}^{T} \left(\hat{R}_t - V_\phi(s_t)\right)^2$
    计算 $\nabla_\phi L_{\text{基线}}$
    将关于 ϕ 的梯度发送至主节点
    等待梯度被接受或丢弃；更新参数
  end for
  if $\text{KL}[\pi_{\text{旧}}||\pi_\theta] > \beta_{\text{高}}\text{KL}_{\text{目标}}$ **then**
    $\lambda \leftarrow \tilde{\alpha}\lambda$
  else if $\text{KL}[\pi_{\text{旧}}||\pi_\theta] < \beta_{\text{低}}\text{KL}_{\text{目标}}$ **then**
    $\lambda \leftarrow \lambda/\tilde{\alpha}$
  end if
end for

们还关注该算法能否扩展到大量工作器，从而在需要大量数据点以获得可靠梯度估计的实验中加速流程。为此，我们在需计算参数更新的样本数量较多的场景下（$N = 100000$）将其与 TRPO 对比。对于简单任务，我们认为 TRPO 在此场景下能产生良好结果（对于基准任务，更小的 N 可能已足够）。

对于 DPPO，我们对策略与基线的学习率进行了粗略搜索。在第 2.1 节中的所有实验使用相同学习率（分别为 0.00005 和 0.0001）。每次迭代中，我们使用的批次大小为：步行器 64000 个时间步、人形机器人 128000 个时间步、抓取器 24000 个时间步。数据采集与梯度计算分布在不同数量的工作器上。由于提前终止，工作器数量有时会更少（当回合提前终止时，当前展开窗口剩余的 K 个时间步在梯度计算中会被忽略）。另一种对比方式是使用固定的总时间步数，同时改变每个工

作器的时间步数。

网络采用 tanh 非线性激活函数，对条件高斯分布的均值与标准差进行参数化以生成动作。网络规模如下：平面步行器为 300,200；人形机器人为 300,200,100；记忆抓取器为 200；100 个 LSTM 单元。

对于连续动作的 A3C，我们同样对相关超参数（尤其是学习率与熵损失）进行了粗略搜索。由于代码库的差异，其网络架构与 DPPO 所用架构并非完全一致，但隐藏单元数量相同。

我们注意到，对这些算法进行一一对应的对比较为困难：它们基于不同的代码库实现，且对于分布式算法，实际运行时间同时受算法的概念性修改与实现选择的影响。对近期几种高吞吐量算法进行更细致的基准测试将是未来的工作方向。

B 额外实验细节

B.1 观测信息

对于所有轨道，地形高度（及适用时的平台高度）以高度场形式提供：每个“像素”表示小区域内的地形（平台）高度。该高度场会在相对于智能体位置的特定点处采样。

平面步行器 平面步行器的外感受特征由地形的采样点（及适用时的平台高度）组成。沿 x 轴有 50 个等距采样点，从智能体后方 2m 处延伸至前方 8m 处。平台高度通过单独的采样点集与地形高度区分开。此外，外感受特征还包含步行器身体距地面的高度（在当前位置测量），以及智能体位置与下一个采样网格中心的差值（这一输入的目的是解决由分段恒定地形表示与有限采样导致的混叠问题）。

四足机器人与人形机器人 四足机器人与人形机器人使用相同的外感受特征集，本质上是步行器所用特征的二维版本。采样点位于可变分辨率网格上：沿 x 轴从智能体后方 1.2m 延伸至前方 5.6m，左右各 4m。为降低输入数据的维度，采样密度随与身体位置的距离增加而降低。除高度采样外，外感受特征还包含身体距地面的高度，以及步行器身体到下一个采样网格中心的 x、y 方向距离（用于减少混叠，见上文）。

B.2 奖励函数

平面步行器

$$r = 10.0v_x + 0.5n_z - |\Delta_h - 1.2| - 10.0\mathcal{I}[\Delta_h < 0.3] - 0.1\|u\|^2$$

其中，n_z 是躯干坐标系 z 轴在全局坐标系 z 轴上的投影（该值介于 1.0 到-1.0 之间，取决于平面步行器的躯干是直立还是倒置）；Δ_h 是平面步行器躯干距足部的高度；$\mathcal{I}[\cdot]$ 是指示函数；v_x 是 x 轴方向的速度。

四足机器人

$$r = v_x + 0.05 n_z - 0.01 \|u\|^2$$

其中，n_z 是躯干坐标系 z 轴在全局坐标系 z 轴上的投影（该值介于 1.0 到-1.0 之间，取决于四足机器人是直立还是倒置）。

人形机器人

$$r = \min(v_x, v_{\text{max}}) - 0.005(v_x^2 + v_y^2) - 0.05 y^2 - 0.02\|u\|^2 + 0.02$$

其中，v_{max} 是速度奖励的截断值，我们通常设为 4m/s。

使用克罗内克分解近似的深度强化学习可扩展信赖域方法[1,2]

摘要

在本文工作中，我们提出将信赖域优化（trust region optimization）应用于深度强化学习，并采用近期提出的克罗内克分解（Kronecker-factored）曲率近似方法。我们扩展了自然策略梯度（natural policy gradient）框架，提出通过带信赖域的克罗内克分解近似曲率（Kronecker-factored approximate curvature, K-FAC）对智能体（actor）和评论家（critic）进行联合优化；因此，我们将所提方法命名为基于克罗内克分解信赖域的智能体-评价家方法（Actor Critic using Kronecker-Factored Trust Region, ACKTR）。据我们所知，该方法是首个适用于智能体-评价家（actor-critic）框架的可扩展信赖域自然梯度方法。同时，该方法能够直接从原始像素输入中学习连续控制与离散控制策略下的非平凡任务（non-trivial tasks）。我们在离散领域（Atari 游戏）与连续领域（MuJoCo 环境）中对所提方法进行了测试。结果表明，与此前最优的在线策略（on-policy）智能体-评价家方法相比，所提方法不仅能获得更高的奖励值（rewards），样本效率（sample effffciency）平均还能提升 2 至 3 倍。本文方法的代码已开源，获取地址为：https://github.com/openai/baselines

1 引言

采用深度强化学习（deep reinforcement learning, deep RL）方法的智能体，已在高维原始感知状态空间中成功习得复杂行为技能，并解决了具有挑战性的控制任务 [25,18,13]。深度强化学习方法借助深度神经网络来表示控制策略。尽管已取得令人瞩目的成果，但这些神经网络仍采用随机梯度下降（stochastic gradient descent, SGD）的简单变体进行训练。随机梯度下降及相关的一阶方法在权重空间中的探索效率较低。当前的深度强化学习方法通常需要数天时间，才能掌握各类连续与离散控制任务。此前有研究提出一种分布式方法 [18]，该方法通过让多个智能体同时与环境交互来缩短训练时间；但随着并行度的提高，该方法在样本效率上的收益会迅速递减。

[1]原文：Yuhuai Wu, Elman Mansimov, Shun Liao, et al. Scalable trust-region method for deep reinforcementlearning using Kronecker-factored approximation [J]. arXiv:1708.05144v2.

[2]译者：徐阳阳。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

样本效率（sample effffciency）是强化学习（RL）中的核心关注点；机器人与真实世界的交互通常比计算时间更稀缺，即便在仿真环境中，仿真成本往往也高于算法本身的成本。有效减少样本量的一种方法是，采用更先进的优化技术来执行梯度更新。自然策略梯度（natural policygradient）[11] 便是采用自然梯度下降（natural gradient descent）[1] 技术来实现梯度更新的。

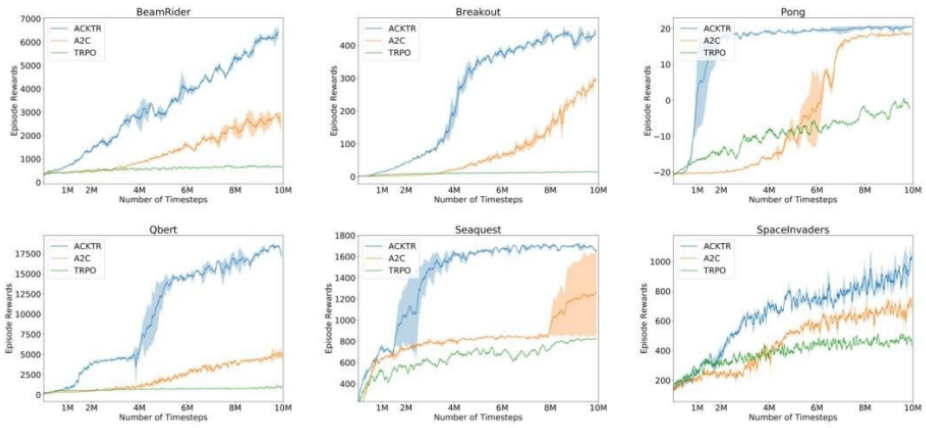


图 1：在六款标准 Atari 游戏上训练 1000 万步（1 步相当于 4 帧）的性能比较. 阴影区域表示基于 2 个随机种子的标准差.

自然梯度方法（natural gradient methods）遵循以费雪度量（Fisher metric）为基础度量的最速下降方向，该度量不依赖于坐标的选择，而是基于流形（即曲面，manifold）本身。

然而，自然梯度的精确计算难以实现，因为这需要对费雪信息矩阵（Fisher information matrix）求逆。信赖域策略优化（Trust-regionpolicy optimization, TRPO）[22] 通过采用费雪向量积（Fisher-vector products）[21]，避免了显式存储和求逆费雪矩阵的操作。但该方法通常需要执行多步共轭梯度（conjugate gradient）计算才能完成一次参数更新，且精确估计曲率（curvature）需要在每个批次中使用大量样本；因此，TRPO 对于大型模型而言并不实用，且存在样本效率低的问题。

Kronecker 分解近似曲率（Kronecker-factored approximated curvature, K-FAC）[16,7] 是对自然梯度（natural gradient）的一种可扩展近似方法。研究表明，在监督学习中，通过使用更大的小批量数据（mini-batches），K-FAC 能够加速各类最先进（state-of-the-art）大规模神经网络 [2] 的训练过程。与信任区域策略优化（TRPO）不同，K-FAC 的每次更新成本与随机梯度下降（SGD）更新相当，并且会对曲率信息（curvature information）进行滑动平均（runningaverage），这使其能够使用小规模批量数据。这一特性表明，将 K-FAC 应用于策略优化（policyoptimization），有望提升当前深度强化学习（deep RL）方法的样本效率（sample efficiency）。

在本文中，我们提出了基于 Kronecker 分解信任区域的演员-评论家算法（actor-critic usingKronecker-factored trust region, ACKTR，发音同“actor”）——一种适用于演员-评论家（actor-critic）方法的可扩展信任区域优化算法。该算法采用 Kronecker 分解近似自然策略梯度（naturalpolicy gradient），使得梯度的协方差矩阵（covariance matrix of the gradient）能够被高效求逆。据我们所知，我们也是首个通过高斯-牛顿近似（Gauss-Newton approximation）将自然策略梯度算法扩展到价值函数（value functions）优化的研究团队。在实际应用中，ACKTR 每次更新的计算成本仅比基于随机梯度下降（SGD）的方法高

本文所用源代码已在线公开，获取地址为：https://github.com/openai/baselines

2 背景

2.1 强化学习与演员-评论家方法

我们考虑智能体与无限时域折扣马尔可夫决策过程 (X, A, γ, P, r) 进行交互。在时刻 t，智能体根据当前状态 $s_t \in X$，按照其策略 $\pi_\theta(a|s_t)$ 选择动作 $a_t \in A$。环境随后产生奖励 $r(s_t, a_t)$，并根据转移概率 $P(s_{t+1}|s_t, a_t)$ 转移到下一状态 s_{t+1}。智能体的目标是针对策略参数 θ 最大化期望 γ-折扣累积回报：$J(\theta) = \mathbb{E}_\pi[R_t] = \mathbb{E}_\pi\left[\sum_{i\geq 0}\gamma^i r(s_{t+i}, a_{t+i})\right]$ 策略梯度方法 [30,26] 直接对策略 $\pi_\theta(a|s_t)$ 进行参数化，并更新参数 θ 以最大化目标函数 $J(\theta)$。其一般形式的策略梯度定义为 [23]：

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

其中 Ψ_t 通常被选为优势函数 $A^\pi(s_t, a_t)$，该函数可提供给定状态 s_t 下每个动作 a_t 的相对价值度量。目前已有大量研究 [23] 致力于设计同时具备低方差与低偏差梯度估计的优势函数。由于这并非本文工作的重点，我们直接遵循异步优势演员-评论家（A3C）方法 [18]，将优势函数定义为带函数近似的 k 步回报，即：

$$A^\pi(s_t, a_t) = \sum_{i=0}^{k-1}\gamma^i r(s_{t+i}, a_{t+i}) + \gamma^k V_\phi^\pi(s_{t+k}) - V_\phi^\pi(s_t)$$

其中 $V_\phi^\pi(s_t)$ 为价值网络，其作用是估计从给定状态出发、遵循策略 π 时的期望累积奖励，即 $V_\phi^\pi(s_t) = \mathbb{E}_\pi[R_t]$。为训练价值网络的参数，我们同样参考 [18] 的方法，通过时序差分更新最小化自举 k 步回报 $\hat{R}_t$ 与预测值之间的平方差，目标函数为：$\frac{1}{2}\left\|\hat{R}_t - V_\phi^\pi(s_t)\right\|^2$

2.2 自然梯度的 Kronecker 分解近似

为了最小化非凸函数 $J(\theta)$，最速下降法计算更新量 $\Delta\theta$ 以最小化 $J(\theta+\Delta\theta)$，其约束条件为 $\|\Delta\theta\|_B < 1$，其中 $\|\cdot\|_B$ 是由 $\|x\|_B = (x^T B x)^{1/2}$ 定义的范数，且 B 是半正定矩阵。该约束优化问题的解形式为 $\Delta\theta \propto -B^{-1}\nabla_\theta J$，其中 $\nabla_\theta J$ 是标准梯度。当范数为欧几里得范数 (即 $B = I$) 时，这就成为常用的梯度下降法。然而，这种变化的欧几里得范数依赖于参数化 θ。这一点并不理想，因为模型的参数化是一种任意选择，不应影响优化轨迹。自然梯度法利用 Fisher 信息矩阵 F (KL 散度的局部二次近似) 来构造范数。在概率分布类上，该范数与模型参数化 θ 无关，从而提供更稳定且有效的更新。但由于现代神经网络可能包含数百万个参数，计算和存储精确的 Fisher 矩阵及其逆矩阵并不现实，因此我们不得不采用近似方法。

$$\min_{\Delta\theta} J(\theta+\Delta\theta)$$

$$\text{s.t. } \Delta\theta^T B \Delta\theta \le 1$$

$$\mathcal{L}(\Delta\theta,\lambda) = \nabla_\theta J^T \Delta\theta + \lambda\left(\Delta\theta^T B\Delta\theta - 1\right)$$

$$\frac{\partial\mathcal{L}}{\partial\Delta\theta} = \nabla_\theta J + 2\lambda B\Delta\theta = 0$$

$$\Delta\theta = -\frac{1}{2\lambda}B^{-1}\nabla_\theta J$$

$$\Delta\theta \propto -B^{-1}\nabla_\theta J$$

近期提出的一种名为Kronecker分解近似曲率（K-FAC）的技术[16]，采用Kronecker分解对 Fisher 矩阵进行近似，以实现高效的近似自然梯度更新。设 $p(y|x)$ 表示神经网络的输出分布，$L = \log p(y|x)$ 表示对数似然。令 $W \in \mathbb{R}^{C_{\text{out}}\times C_{\text{in}}}$ 为第 ℓ 层的权重矩阵，其中 C_{out} 和 C_{in} 分别表示该层的输出神经元数量与输入神经元数量。记该层的输入激活向量为 $a \in \mathbb{R}^{C_{\text{in}}}$，下一层的预激活向量为 $s = Wa$。注意，权重梯度可表示为 $\nabla_W L = (\nabla_s L)a^\top$。

$$s_k = \sum_{t=1}^{C_{\text{in}}} W_{kt}a_t \quad (\forall k = 1,\ldots,C_{\text{out}})$$

$$\frac{\partial L}{\partial W_{ij}} = \sum_{k=1}^{C_{\text{out}}} \frac{\partial L}{\partial s_k}\cdot\frac{\partial s_k}{\partial W_{ij}}$$

$$\frac{\partial s_k}{\partial W_{ij}} = \begin{cases} a_j & k = i \\ 0 & k \ne i \end{cases}$$

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial s_i} \cdot a_j$$

$$\nabla_s L = \left[\frac{\partial L}{\partial s_1}, \cdots, \frac{\partial L}{\partial s_{C_{\text{out}}}}\right]^T$$

$$(\nabla_s L) a^T \text{ 的} (i,j) \text{ 元素: } \frac{\partial L}{\partial s_i} \cdot a_j$$

$$\nabla_W L \text{ 的} (i,j) \text{ 元素: } \frac{\partial L}{\partial W_{ij}}$$

$$\nabla_W L = (\nabla_s L) a^T$$

K-FAC 利用这一特性，并进一步将第 ℓ 层对应的块 F_ℓ 近似为 $\hat{F}_\ell$，具体如下：

$$\begin{aligned} F_\ell &= \mathbb{E}\left[\text{vec}\{\nabla_W L\}\text{vec}\{\nabla_W L\}^\top\right] = \mathbb{E}\left[aa^\top \otimes \nabla_s L(\nabla_s L)^\top\right] \\ &\approx \mathbb{E}\left[aa^\top\right] \otimes \mathbb{E}\left[\nabla_s L(\nabla_s L)^\top\right] := A \otimes S := \hat{F}_\ell, \end{aligned}$$

其中 A 表示 $\mathbb{E}\left[aa^\top\right]$，$S$ 表示 $\mathbb{E}\left[\nabla_s L(\nabla_s L)^\top\right]$。该近似可理解为假设激活值的二阶统计量与反向传播导数的二阶统计量互不相关。借助这一近似，利用基本恒等式 $(P \otimes Q)^{-1} = P^{-1} \otimes Q^{-1}$ 和 $(P \otimes Q)\text{vec}(T) = \text{vec}(PTQ^\top)$，可高效计算自然梯度更新：

$$\text{vec}(\Delta W) = \hat{F}_\ell^{-1}\text{vec}\left\{\nabla_W J\right\} = \text{vec}\left(A^{-1}\nabla_W J S^{-1}\right).$$

由上式可见，K-FAC 近似自然梯度更新仅需对与 W 尺寸相当的矩阵进行计算。Grosse 与 Martens[?] 近期已将 K-FAC 算法扩展至卷积网络。随后，Ba 等人[?] 提出了该方法的分布式版本，通过异步计算大幅降低了大部分开销。分布式 K-FAC 在训练大型现代分类卷积网络时，实现了 2 至 3 倍的训练速度提升。

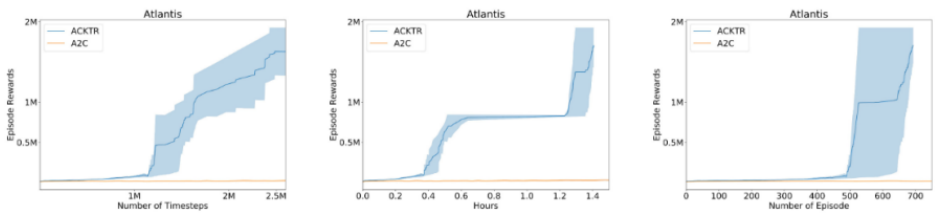


图 2：在雅达利游戏《亚特兰蒂斯》中，我们的智能体（ACKTR）在 1.3 小时、600

局游戏、250 万个时间步内迅速学会获得 200 万的奖励。相同的结果由优势演员评论家（A2C）在 10 小时、6000 局游戏、2500 万个时间步内实现。在该游戏中，ACKTR 的样本效率是 A2C 的 10 倍。

3 方法

3.1 兼容函数近似

十多年前，Kakade[11] 就提出将自然梯度应用于策略梯度方法。但至今仍缺乏一种可扩展、样本高效且通用的自然策略梯度实现方案。在本节中，我们将介绍首个适用于演员-评论家方法的可扩展且样本高效的自然梯度算法：基于 Kronecker 分解信任域的演员-评论家 (ACKTR) 方法。该方法采用 Kronecker 分解近似计算自然梯度更新，并将自然梯度更新同时应用于演员网络与评论家网络。

为定义强化学习目标的 Fisher 度量，一个自然的选择是利用策略函数 (该函数定义了给定当前状态下动作的概率分布)，并对轨迹分布求期望：

$$F = \mathbb{E}_{p(\tau)}\left[\nabla_\theta \log \pi(a_t|s_t)\left(\nabla_\theta \log \pi(a_t|s_t)\right)^\top\right],$$

其中 $p(\tau)$ 为轨迹分布，其表达式为 $p(s_0)\prod_{t=0}^{T}\pi(a_t|s_t)p(s_{t+1}|s_t,a_t)$。在实际应用中，通常通过训练过程中收集的轨迹来近似这一难以直接计算的轨迹期望。

接下来，我们将介绍一种将自然梯度应用于评论家网络优化的方法。评论家网络的学习可视为一个最小二乘函数近似问题，尽管其目标值会动态变化。在最小二乘函数近似场景中，常用的二阶算法是高斯-牛顿法，该方法将曲率近似为高斯-牛顿矩阵 $G := \mathbb{E}[J^\top J]$，其中 J 是从参数到输出的映射的雅可比矩阵 [19]。对于高斯观测模型，高斯-牛顿矩阵与 Fisher 矩阵等价 [15]；这种等价性使得我们同样可将 K-FAC 应用于评论家网络。具体而言，我们假设评论家网络的输出 v 服从高斯分布 $p(v|s_t) \sim \mathcal{N}(v; V(s_t), \sigma^2)$。评论家网络的 Fisher 矩阵即基于该高斯输出分布定义。在实际操作中，可直接将 σ 设为 1，这与标准高斯-牛顿法等价。

若演员网络与评论家网络相互独立，可利用上述定义的度量分别对两个网络进行 K-FAC 更新。但为避免训练不稳定性，采用“两个网络共享底层表示、但拥有独立输出层”的架构往往更有利 [18,28]。在这种情况下，我们可通过假设两个输出分布相互独立来定义策略与价值分布的联合分布，即 $p(a,v|s) = \pi(a|s)p(v|s)$，并基于 $p(a,v|s)$ 构造 Fisher 度量。该过程与标准 K-FAC 基本一致，仅需独立对两个网络的输出进行采样。随后，我们可应用 K-FAC 近似 Fisher 矩阵 $\mathbb{E}_{p(\tau)}[\nabla \log p(a,v|s)\nabla \log p(a,v|s)^\top]$，从而对两个网络进行同步更新。

此外，我们采用文献 [16] 中提出的因子化吉洪诺夫阻尼方法。同时，为缩短

计算时间，我们还遵循文献 [2] 的思路，对 Kronecker 近似所需的二阶统计量与逆矩阵进行异步计算。

3.2 步长选择与信赖域优化

传统上，自然梯度采用类似 SGD 的更新方式：$\theta \leftarrow \theta - \eta F^{-1}\nabla_\theta L$。但在深度强化学习背景下，Schulman 等人 [22] 发现，这种更新规则可能导致对策略的大幅更新，使算法过早收敛到接近确定性的策略。因此，他们主张采用信任域方法，即将更新幅度缩小，以确保策略分布的变化 (以 KL 散度衡量) 不超过指定量。

为此，我们采用文献 [2] 提出的 K-FAC 信任域形式，将有效步长 η 选择为 $\sqrt{\min\left(\eta_{\max}, \frac{2\delta}{\Delta\theta^\top \hat{F}\Delta\theta}\right)}$，其中学习率 $\eta_{\max}$ 和信任域半径 δ 是超参数。若演员网络与评论家网络相互独立，则需要为两者分别调整不同的 $\eta_{\max}$ 和 δ。评论家输出分布的方差参数可融入标准高斯-牛顿法的学习率参数中。另一方面，若两者共享表示，则需要调整一组 $\eta_{\max}$、δ，以及评论家与演员训练损失的权重参数。

4 相关工作

自然梯度 [1] 最早由 Kakade[11] 应用于策略梯度方法。Bagnell 和 Schneider[3] 进一步证明 [11] 中定义的度量是由路径分布流形引起的协变度量。Peters 和 Schaal[20] 随后将自然梯度应用于演员-批评算法。他们建议对参与者的更新执行自然策略梯度，并使用最小二乘时间差 (LSTD) 方法对批评者的更新进行更新。然而，在应用自然梯度方法时存在很大的计算挑战，主要与有效存储费舍尔矩阵以及计算其逆矩阵有关。为了易于处理，之前的工作将该方法限制为使用兼容的函数近似器 (线性函数近似器)。为了避免计算负担，信任区域策略优化 (TRPO)[22] 使用共轭梯度与快速 Fisher 矩阵-向量乘积近似求解线性系统，类似于 Martens[14] 的工作。这种方法有两个主要缺点。首先，它需要重复计算 Fisher 矢量乘积，从而阻止其扩展到 Atari 和 MuJoCo 中从图像观察中学习的实验中通常使用的更大架构。其次，它需要大量推出才能准确估计曲率。K-FAC 通过使用易于处理的 Fisher 矩阵近似值并在训练期间保持曲率统计的运行平均值来避免这两个问题。尽管 TRPO 比使用一阶优化器 (如 Adam) 训练的策略梯度方法 (如 Adam[12]) 显示出更好的每次迭代进度，但它的样本效率通常较低。

提出了几种方法来提高 TRPO 的计算效率。为了避免重复计算费舍尔向量积，Wang 等人 [28] 使用策略网络运行平均与当前策略网络之间 KL 散度的线性近似来解决约束优化问题。Heess 等人 [9] 和 Schulman 等人 [24] 没有采用信任域优化器施加的硬约束，而是在目标函数中添加了 KL 代价作为软约束。两篇论文都显示，

在连续和离散控制任务中，在样本效率方面相比普通策略梯度有所提升。

还有其他最近提出的演员-评论家模型，通过引入经验回放 [28],[8] 或辅助目标 [10] 来提高样本效率。这些方法与我们的工作是正交的，并且可能与 ACKTR 结合进一步提高样本效率。

5 实验

我们进行了系列实验以研究以下问题：(1) 在样本效率和计算效率方面，ACKTR 与最先进的在策略方法和常见的二阶优化器基线相比表现如何？(2) 对于评论者的优化，哪种范数更好？(3) 与一阶方法相比，ACKTR 的性能随批量大小如何变化？

表 1：ACKTR 和 A2C 结果显示了在 5 千万时间步后获得的最后 100 个平均回合奖励，以及 TRPO 在 1000 万时间步后的结果。表中还显示了回合 N，其中 N 表示第一个游戏的回合，在该回合中，从第 N 局到第 (N+100) 局的平均回合奖励超过了人类表现水平 [17]，结果是基于两个随机种子的平均值。

我们在两个标准基准平台上评估了我们提出的方法 ACKTR。我们首先在 OpenAI Gym [5] 中定义的离散控制任务上进行了评估，这些任务由 Arcade Learning Environment [4] 模拟，这是一款用于 Atari 2600 游戏的模拟器，通常作为离散控制的深度强化学习基准。我们随后对其在多种连续控制基准任务上进行了评估，这些任务定义于 OpenAI Gym 环境 [5]，并由 MuJoCo 物理引擎 [27] 模拟生成。我们选用的基准方法包括：（a）异步优势演员-评论员模型（A3C）[18] 的异步批量版本，此后简称 A2C（优势演员-评论员）；（b）信任区域策略优化（TRPO）[22]。ACKTR 与所有基准方法采用相同的模型架构，但在 Atari 游戏任务上的 TRPO 基准方法除外。由于运行共轭梯度内循环会带来计算负担，因此在该任务场景下，我们只能为 TRPO 采用更小的模型架构。其他实验细节详见附录。

5.1 离散控制

我们首先展示了标准的六款 Atari 2600 游戏的结果，以衡量 ACKTR 带来的性能提升。图 1 显示了在 1,000 万时间步训练的六款 Atari 游戏的结果，并与 A2C 和 TRPO 进行了比较。ACKTR 在样本效率 (即每个时间步的收敛速度) 方面显著优于 A2C，在所有游戏中都有明显优势。我们发现 TRPO 在 1,000 万时间步内只能学习两款游戏——《深海探险》和《乒乓》，且其样本效率表现不如 A2C。

Domain	Human level	ACKTR		A2C		TRPO (10 M)	
		Rewards	Episode	Rewards	Episode	Rewards	Episode
Beamrider	5775.0	**13581.4**	**3279**	8148.1	8930	670.0	N/A
Breakout	31.8	**735.7**	**4094**	581.6	14464	14.7	N/A
Pong	9.3	**20.9**	**904**	19.9	4768	-1.2	N/A
Q-bert	13455.0	**21500.3**	**6422**	15967.4	19168	971.8	N/A
Seaquest	**20182.0**	1776.0	N/A	1754.0	N/A	810.4	N/A
Space Invaders	1652.0	**19723.0**	**14696**	1757.2	N/A	465.1	N/A

在表 1 中，我们展示了在训练 50 百万时间步中最后 100 个回合的平均奖励，以及达到人类水平所需的回合数 [17]。值得注意的是，在游戏 Beamrider、Breakout、Pong 和 Q-bert 中，A2C 分别比 ACKTR 需要多 2.7、3.5、5.3 和 3.0 倍的回合才能达到人类水平。此外，A2C 在 Space Invaders 的一个运行中未能达到人类表现，而 ACKTR 的平均得分为 19723，比人类表现 (1652) 高 12 倍。在游戏 Breakout、Q-bert 和 Beamrider 中，ACKTR 比 A2C 取得了 26

我们还评估了 ACKTR 在其他 Atari 游戏上的表现；完整结果见附录 B。我们将 ACKTR 与 Q 学习方法进行了比较，发现 ACKTR 在 44 个基准测试中有 36 个在样本效率上与 Q 学习方法相当。

5.2 连续控制

我们在 OpenAI Gym[5] 定义的连续控制任务标准基准上进行了实验，这些任务在 MuJoCo[27] 中模拟，包括低维状态空间表示和直接从像素输入的情形。与 Atari 相比，连续控制任务有时更具挑战性，因为动作空间高维且需要探索。图 3 显示了在八个 MuJoCo 环境中训练 1 百万时间步的结果。我们的模型在八个 MuJoCo 任务中有六个明显超过基线，并在另外两个任务 (Walker2d 和 Swimmer) 上与 A2C 表现相当。

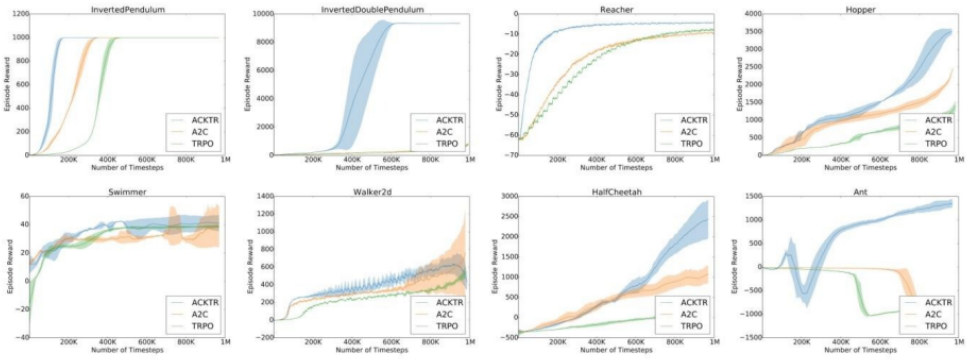


图 3：在八个 MuJoCo 环境上训练 100 万时间步（1 个时间步等于 4 帧）的性能比较。阴影区域表示 3 个随机种子的标准偏差

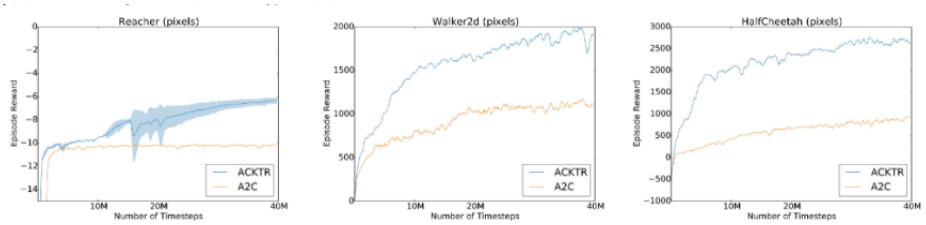


图 4：在 3 个 MuJoCo 环境中基于图像观察的性能比较，训练了 4000 万时间步（1 个时间步等于 4 帧）。

我们进一步在八个 MuJoCo 任务上对 ACKTR 进行了 3000 万时间步的评估，在表 2 中我们展示了训练中前 10 个连续回合的平均奖励，以及数量达到文献 [8] 中定义的某个阈值所需的训练集次。如表 2 所示，ACKTR 在除 Swimmer 外的所有任务上都能更快达到指定阈值，其中在 Swimmer 任务中 TRPO 的样本效率是其 4.1 倍。一个特别显著的例子是 Ant 任务，其中 ACKTR 的样本效率比 TRPO 高 16.4 倍。至于平均奖励分数，除 TRPO 在 Walker2d 环境中取得 10 更高的奖励分数外，三种模型的结果相互之间都相当。

我们还尝试从像素直接学习连续控制策略，而不提供低维状态空间作为输入。从像素学习连续控制策略比从状态空间学习要困难得多，部分原因是渲染速度比 Atari 慢 (MuJoCo 为 0.5 秒，而 Atari 为 0.002 秒)。最先进的演员-评论家方法 A3C[18] 仅在相对简单的任务上，如 Pendulum、Pointmass2D 和 Gripper，报告了从像素学习的结果。如图 4 所示，我们可以看到，在训练 4000 万步之后，我们的模型在最终回合奖励方面显著优于 A2C。更具体地说，在 Reacher、HalfCheetah 和 Walker2d 上，我们的模型相比 A2C 实现了 1.6、2.8 和 1.7 倍更高的最终奖励。从像素训练的策略视频可以在`https://www.youtube.com/watch?v=gtM87w1xGoM`查看。预训练模型权重可在`https://github.com/emansim/acktr`获取

5.3 批评优化的更好标准?

之前的自然策略梯度方法仅对演员应用自然梯度更新。在我们的工作中，我们还提出将自然梯度更新应用于评论家。区别在于我们选择用于在评论家上执行最速下降的范数；即在第 2.2 节中定义的范数 $\|\cdot\|_B$。在本节中，我们对演员应用了 ACKTR，并比较了使用一阶方法 (即欧几里得范数) 与使用 ACKTR(即由高斯-牛顿定义的范数) 进行评论家优化的效果。图 5(a) 和 (b) 显示了在连续控制任务 HalfCheetah 和 Atari 游戏 Breakout 上的结果。我们观察到，无论使用哪种范数来优化评论家，将 ACKTR 应用于演员相比基线 A2C 都有改进。

		ACKTR		A2C		TRPO (10 M)	
Domain	Threshold	Rewards	Episodes	Rewards	Episodes	Rewards	Episodes
Ant	3500 (6000)	4621.6	**3660**	4870.5	106186	**5095.0**	60156
HalfCheetah	4700 (4800)	5586.3	**12980**	5343.7	21152	**5704.7**	21033
Hopper	2000 (3800)	**3915.9**	**17033**	3915.3	33481	3755.0	39426
InvertedPendulum	950 (950)	1000.0	**6831**	1000.0	10982	1000.0	29267
InvertedDoublePendulum	9100 (9100)	9356.0	**41996**	**9356.1**	82694	9320.0	78519
Reacher	-7 (-3.75)	**-1.5**	**3325**	-1.7	20591	-2.0	14940
Swimmer	90 (360)	138.0	6475	**140.7**	11516	136.4	**1571**
Walker2d	3000 (N/A)	6198.8	**15043**	5874.9	26828	**6874.1**	27720

表 2：ACKTR、A2C 和 TRPO 的结果，显示在 3000 万时间步内获得的前 10 名平均回合奖励，这些结果是对 8 个随机种子中表现最好的 3 个种子的平均值。“回合”表示最小的 N，在该 N 到第 (N+10) 个游戏中平均回合奖励超过某个阈值。除倒立摆和倒立双摆外，所有环境的阈值均按照 Gu 等人 [8] 选择，括号中显示的是根据 OpenAI Gym 网站 [5] 解决该环境所需的奖励阈值。

然而，使用高斯-牛顿范数优化关键点所带来的改进在样本效率和训练结束时的回报方面更为显著。此外，我们观察到使用欧几里得范数时结果在随机种子上存在较大差异，而高斯-牛顿范数则有助于稳定训练。

回顾可知，评价者的费希尔矩阵是利用评价者的输出分布 (方差为 σ 的高斯分布) 构建的。在标准高斯-牛顿法中，σ 被设为 1。我们尝试使用贝尔曼误差的方差来估计 σ，这类似于回归分析中对噪声方差的估计，我们将这种方法称为自适应高斯-牛顿法。然而，我们发现自适应高斯-牛顿法与标准高斯-牛顿法相比没有任何显著改进 (关于 σ 选择的详细比较见附录 D)。

5.4 ACKTR 与 A2C 在实际耗时上比较如何?

我们将 ACKTR 与基准算法 A2C 和 TRPO 在实际耗时 (wall-clock time) 方面进行了对比。表 3 展示了在 6 个 Atari 游戏和 8 个 MuJoCo(基于状态空间) 环境中，每秒处理的平均时间步长。该结果是在与之前实验相同的实验设置下获得的。需要注意的是，MuJoCo 任务中的回合是按顺序处理的，而 Atari 环境中的回合则是并行处理的；因此 Atari 环境中能处理更多帧数据。从表中可以看出，ACKTR 每时间步仅增加了最多 25 的计算时间，这表明它在具备显著优化收益的同时，还具有良好的实用性

(Timesteps/Second)	Atari			MuJoCo		
batch size	80	160	640	1000	2500	25000
ACKTR	712	753	852	519	551	582
A2C	1010	1038	1162	624	650	651
TRPO	160	161	177	593	619	637

表 3：计算成本比较。在训练过程中，每种算法在六个 Atari 游戏和八个 MuJoCo 任务上的平均每秒时间步数。ACKTR 相对于 A2C 只最多增加 25

5.5 ACKTR 和 A2C 在不同批量大小下表现如何?

在大规模分布式学习环境中，优化通常使用较大的批量大小。因此，在这种环境下，最好使用能够很好适应批量大小的方法。在本节中，我们比较了 ACKTR 和基线 A2C 在不同批量大小下的表现。我们实验了 160 和 640 的批量大小。图 5(c) 显示了以时间步为单位的奖励。我们发现，使用较大批量大小的 ACKTR 表现与使用较小批量大小的相当。然而，A2C 在较大批量大小下的样本效率显著下降。

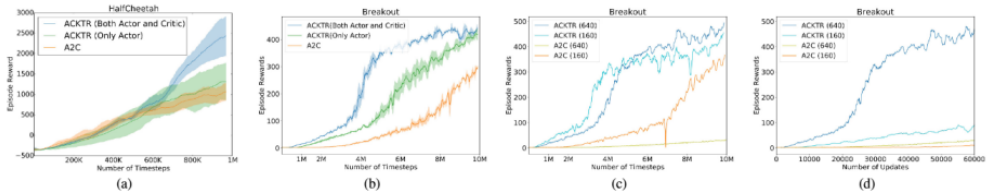


图 5：(a) 和 (b) 对比了使用高斯-牛顿范数（ACKTR）与欧几里得范数（一级方法）优化评价器（价值网络）。(c) 和 (d) 对比了不同批量大小下的 ACKTR 和 A2C

6 讨论

在这项工作中，我们提出了一种样本高效且计算成本低的深度强化学习信任域优化方法。我们使用了一种最近提出的技术——K-FAC，来近似演员-评论家方法的自然梯度更新，并结合信任域优化以保证稳定性。据我们所知，我们是首个提出同时使用自然梯度更新优化演员和评论家的人。我们在 Atari 游戏以及 MuJoCo 环境中测试了我们的方法，发现与一阶梯度方法（A2C）和迭代二阶方法（TRPO）相比，样本效率平均提高了 2 至 3 倍。由于我们算法的可扩展性，我们也是首个能够直接从原始像素观测空间训练多个非平凡连续控制任务的方法。这表明，将 Kronecker 分解自然梯度近似扩展到强化学习中的其他算法是一个有前景的研究方向。

致谢

我们要感谢 OpenAI 团队在提供基线结果和 Atari 环境预处理代码方面的慷慨支持。我们还要感谢 John Schulman 的有益讨论。

参考文献

A 实验细节

A.1 离散控制

在 Atari 环境的实验中，我们采用了与文献 [17] 相同的输入预处理程序，但对网络结构进行了轻微修改。具体来说，我们使用一个共享网络来参数化策略和价值函数：第一个卷积层包含 32 个大小为 8×8、步长为 4 的滤波器，接着是第二个卷积层，包含 64 个大小为 4×4、步长为 2 的滤波器，然后是最后一个卷积层，包含 32 个大小为 3×3、步长为 1 的滤波器，之后是一个大小为 512 的全连接层，最后是一个用于参数化策略的 softmax 输出层和一个用于预测价值的线性输出层。我们在第三个卷积层中使用 32 个滤波器，是因为我们发现这样在计算 Fisher 矩阵逆时可以节省时间，且不会导致性能下降（另一种选择是对所有 64 个滤波器使用双因子近似 [2]）。对于基线 A2C，我们使用了与文献 [17] 相同的结构。对于 TRPO，由于其每次迭代的开销较大，我们使用了一个更小的结构，包含 2 个卷积层和一个具有 128 个单元的全连接层。第一个卷积层有 8 个大小为 8×8、步长为 4 的滤波器，接着是另一个卷积层，有 16 个大小为 4×4、步长为 2 的滤波器。我们在 Breakout 游戏上通过对 0.7, 0.2, 0.07, 0.02 进行网格搜索来调整最大学习率 ηmax，并将信任区域半径 δ 设置为 0.001。所有 Atari 实验都使用相同的超参数。基线方法（A2C）和我们的方法在训练过程中均采用学习率的线性调度，并使用权重为 0.01 的熵正则化。遵循文献 [17]，智能体在每个游戏上使用 5000 万时间步长或 2 亿帧进行训练。除非另有说明，我们对 ACKTR 使用 640 的批处理大小，对 A2C 使用 80 的批处理大小，对 TRPO 使用 512 的批处理大小。选择这些批处理大小是为了获得更好的样本效率。

A.2 连续控制

对于以低维状态空间作为输入的实验，我们使用了两个独立的神经网络，每个两层网络的隐藏层包含 64 个单元。我们分别在策略网络和价值网络的所有层（输出层除外，输出层没有任何非线性）中使用了 Tanh 和 ELU [6] 非线性函数。高斯策略的对数标准差被参数化为策略网络最终一层的偏置，与输入状态无关。在所有实验中，我们使用了批量大小为 2500 的 ACKTR 和 A2C 训练，以及批量大小为 25000 的 TRPO 训练。这些批量大小的选择与 OpenAI 团队提供的实验设计结果保持一致。对于使用像素作为输入的实验，我们将一张 42 × 42 的 RGB 图像以及上一帧输入到卷积神经网络中。两个卷积层包含 32 个 3 × 3 尺寸的滤波器，步幅为 2，后接一个具有 256 个隐藏单元的全连接层。与我们的 Atari 实验相比，我们发现将

策略网络和价值函数分成两个独立的网络，在 ACKTR 和 A2C 中都能得到更好的实际表现。我们为策略网络使用 ReLU 非线性，为价值函数使用 ELU [6] 非线性。我们还发现，对两个网络使用正交初始化非常重要，否则 A2C 基线无法提高其回合奖励。所有模型均以 8000 的批量大小进行训练。我们通过在 Reacher 和 Hopper 任务上对 0.3,0.03,0.003 进行网格搜索来调整最大学习率 ffmax，信赖域半径 ff 设置为 0.001。我们固定了所有 MuJoCo 实验的超参数

GAMES	HUMAN	DQN	DDQN	DUEL.	PRIOR.	PRIOR. DUEL.	ACKTR
Alien	7,127.7	1,620.0	3,747.7	4,461.4	4,203.8	3,941.0	3197.1
Amidar	1,719.5	978.0	1,793.3	2,354.5	1,838.9	2,296.8	1059.4
Assault	742.0	4,280.4	5,393.2	4,621.0	7,672.1	11,477.0	10,777.7
Asterix	8,503.3	4,359.0	17,356.5	28,188.0	31,527.0	375,080.0	31,583.0
Asteroids	47,388.7	1,364.5	734.7	2,837.7	2,654.3	1,192.7	34,171.6
Atlantis	29,028.1	279,987.0	106,056.0	382,572.0	357,324.0	395,762.0	3,433,182.0
Bank Heist	753.1	455.0	1,030.6	1,611.9	1,054.6	1,503.1	1,289.7
Battle Zone	37,187.5	29,900.0	31,700.0	37,150.0	31,530.0	35,520.0	8910.0
Beamrider	16,926.5	8,627.5	13,772.8	12,164.0	23,384.2	30,276.5	13,581.4
Berzerk	2,630.4	585.6	1,225.4	1,472.6	1,305.6	3,409.0	927.2
Bowling	160.7	50.4	68.1	65.5	47.9	46.7	24.3
Boxing	12.1	88.0	91.6	99.4	95.6	98.9	1.45
Breakout	30.5	385.5	418.5	345.3	373.9	366.0	735.7
Centipede	12,017.0	4,657.7	5,409.4	7,561.4	4,463.2	7,687.5	7,125.28
Crazy Climber	35,829.4	110,763.0	117,282.0	143,570.0	141,161.0	162,224.0	150,444.0
Demon Attack	1,971.0	12,149.4	58,044.2	60,813.3	71,846.4	72,878.6	274,176.7
Double Dunk	-16.4	-6.6	-5.5	0.1	18.5	-12.5	-0.54
Enduro	860.5	729.0	1,211.8	2,258.2	2,093.0	2,306.4	0.0
Fishing Derby	-38.7	-4.9	15.5	46.4	39.5	41.3	33.73
Freeway	29.6	30.8	33.3	0.0	33.7	33.0	0.0
Gopher	2,412.5	8,777.4	14,840.8	15,718.4	32,487.2	104,368.2	47,730.8
Ice Hockey	0.9	-1.9	-2.7	0.5	1.3	-0.4	-4.2
James Bond	302.8	768.5	1,358.0	1,312.5	5,148.0	812.0	490.0
Kangaroo	3,035.0	7,259.0	12,992.0	14,854.0	16,200.0	1,792.0	3,150.0
Krull	2,665.5	8,422.3	7,920.5	11,451.9	9,728.0	10,374.4	9,686.9
Kung-Fu Master	22,736.3	26,059.0	29,710.0	34,294.0	39,581.0	48,375.0	34,954.0
Phoenix	7,242.6	8,485.2	12,252.5	23,092.2	18,992.7	70,324.3	133,433.7
Pitfall!	6,463.7	-286.1	-29.9	0.0	-356.5	0.0	-1.1
Pong	14.6	19.5	20.9	21.0	20.6	20.9	20.9
Q-bert	13,455.0	13,117.3	15,088.5	19,220.3	16,256.5	18,760.3	23,151.5
River Raid	17,118.0	7,377.6	14,884.5	21,162.6	14,522.3	20,607.6	17,762.8
Road Runner	7,845.0	39,544.0	44,127.0	69,524.0	57,608.0	62,151.0	53,446.0
Robotank	11.9	63.9	65.1	65.3	62.6	27.5	16.5
Seaquest	42,054.7	5,860.6	16,452.7	50,254.2	26,357.8	931.6	1,776.0
Solaris	12,326.7	3,482.8	3,067.8	2,250.8	4,309.0	133.4	2,368.6
Space Invaders	1,668.7	1,692.3	2,525.5	6,427.3	2,865.8	15,311.5	19,723.0
Star Gunner	10,250.0	54,282.0	60,142.0	89,238.0	63,302.0	125,117.0	82,920.0
Time Pilot	5,229.2	4,870.0	8,339.0	11,666.0	9,197.0	7,553.0	22,286.0
Tutankham	167.6	68.1	218.4	211.4	204.6	245.9	314.3
Up and Down	11,693.2	9,989.9	22,972.2	44,939.6	16,154.1	33,879.1	436,665.8
Video Pinball	17,667.9	196,760.4	309,941.9	98,209.5	282,007.3	479,197.0	100,496.6
Wizard Of Wor	4,756.5	2,704.0	7,492.0	7,855.0	4,802.0	12,352.0	702.0
Yars' Revenge	54,576.9	18,098.9	11,712.6	49,622.1	11,357.0	69,618.1	125,169.0
Zaxxon	9,173.3	5,363.0	10,163.0	12,944.0	10,469.0	13,886.0	17,448.0

表 4：从 30 轮无操作动作开始的所有游戏原始分数。其他分数来自 [29]。

B 剩余 Atari 游戏的结果

在本节中，我们在表 4 中展示了其他雅达利游戏的结果。我们方法报告的分数是经过 5000 万时间步后的最后 100 个回合奖励的平均值。每个回合开始时有 30 个无操作（no-op）动作。我们发现 A3C[18] 或 A2C 没有使用相同指标的结果报告。因此，我们将我们的结果与其他从 [29] 获得的 Q 学习方法进行比较。由于计算资源有限，我们仅能够在部分游戏上评估 ACKTR。我们的结果是使用单个随机种子

获得的，并且没有调优任何超参数。尽管我们只使用一个随机种子，我们的结果仍与使用离策略技术（如经验回放）的 Q 学习方法相当。Q 学习方法通常需要数天才能完成一次训练，而我们的方法在现代 GPU 上仅需 16 小时。

C MuJoCo 结果与 OpenAI 基线的比较

我们将 ACKTR 与 OpenAI 团队提供给我们的 A2C 和 TRPO 结果进行了比较（https://github.com/openai/baselines-results）。我们尽可能遵循了他们的实验方案。与我们的基线方法一样，A2C 和 TRPO 都使用相同的两层架构，每层有 64 个隐藏单元，批量大小分别为 2500 和 25000 进行训练。然而，与我们的基线方法不同，价值函数使用了 Tanh 非线性，并通过计算更新前后价值函数的加权平均值进行“软更新”。与我们实现的 A2C 基线相比，OpenAI 实现的 A2C 在 Hopper、InvertedPendulum、Swimmer 和 Walker2d 任务上表现更好，而在 Reacher 和 HalfCheetah 任务上表现较差。OpenAI 的 TRPO 在 Hopper 上表现不如我们训练的 TRPO，而在其余任务上的表现相同。结果如图 6 所示。

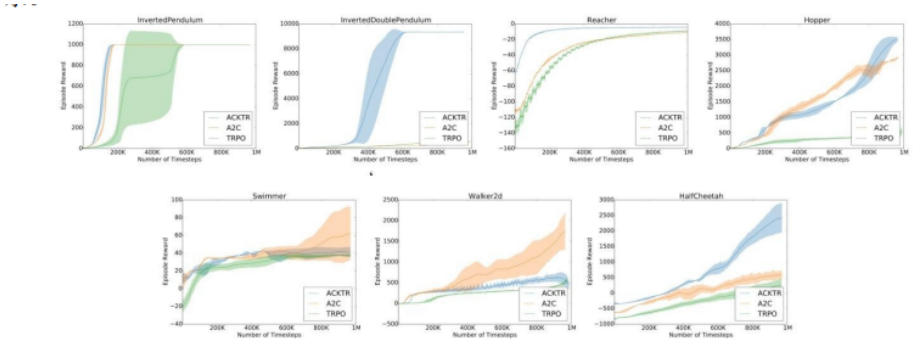


图 6：在七个 MuJoCo 环境中训练 100 万时间步（1 时间步等于 4 帧）的性能比较。A2C 和 TRPO 的数据由 OpenAI 团队提供。阴影区域表示三个随机种子的标准差。

D 自适应高斯-牛顿？

在本节中，我们研究了使用自适应高斯-牛顿训练评论器（即将贝尔曼误差的标准差估计为评论器输出分布的标准差）是否比普通高斯-牛顿（两者在第 3.1 节中定义）提供任何改进。我们在所有六个标准的 Atari 游戏和八个 MuJoCo 任务上运行了自适应高斯-牛顿。结果显示在图 7 和图 8 中。我们看到，在 Atari 游戏中，自适应高斯-牛顿在 Beamrider、Q-bert 和 Seaquest 的样本效率方面降低了性能，在 Pong 游戏中仅显示出轻微的改进。在 MuJoCo 任务中，自适应高斯-牛顿在

InvertedDoublePendulum、Swimmer、Walker2d 和 Ant 任务上略有改进，在 InvertedPendulum 和 Reacher 任务上表现与普通高斯-牛顿相当，而在 HalfCheetah 任务上的表现则明显逊色于普通高斯-牛顿。

E Kronecker 分解的二次近似与精确 KL 匹配得如何？

我们通过在训练过程中测量精确的 KL 变化，同时使用 Kronecker 分解的二次模型进行信赖域优化，间接测试了 Kronecker 分解的曲率近似的准确性。我们在两个 Mujoco 环境中进行了测试，分别是 HalfCheetah 和 Reacher。近似 KL 和精确 KL 的数值如图 9 所示。从图中我们可以看到该精确的 KL 值接近信赖域半径，显示了通过 Kronecker 分解近似进行信赖域优化的有效性。

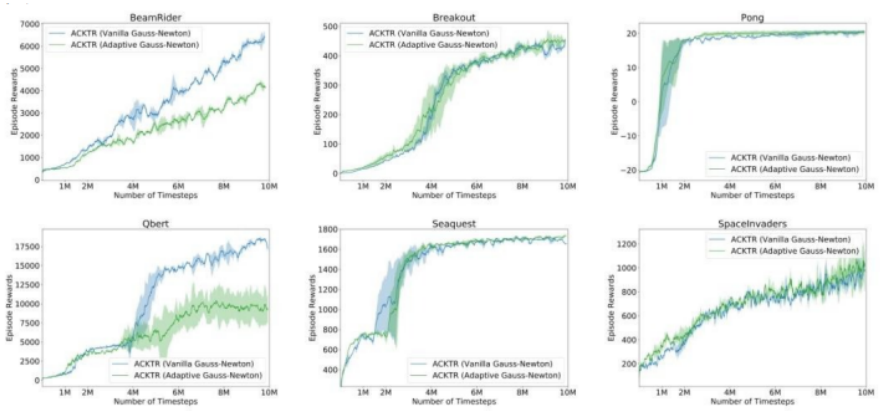


图 7：在六个 Atari 环境中训练 10 百万时间步（1 个时间步等于 4 帧）时，自适应高斯-牛顿和标准高斯-牛顿训练的评论家性能比较。阴影区域表示 2 个随机种子的标准差。

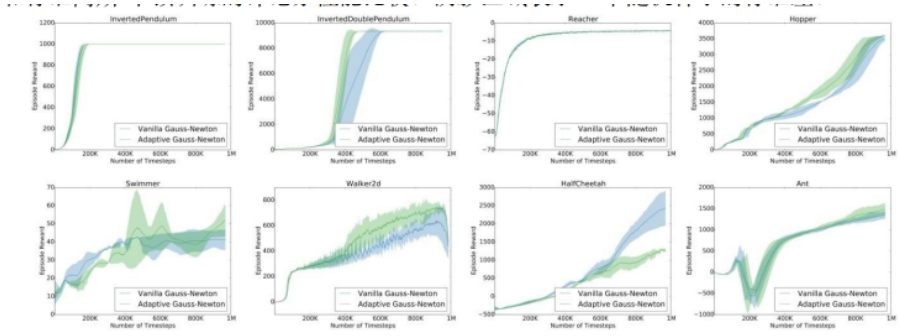


图 8：在八个 MuJoCo 环境中，对使用自适应高斯-牛顿和普通高斯-牛顿训练的评论者性能进行的比较，训练时间为 100 万时间步（1 时间步等于 4 帧）。阴影区域表示 3 个随机种子的标准差。

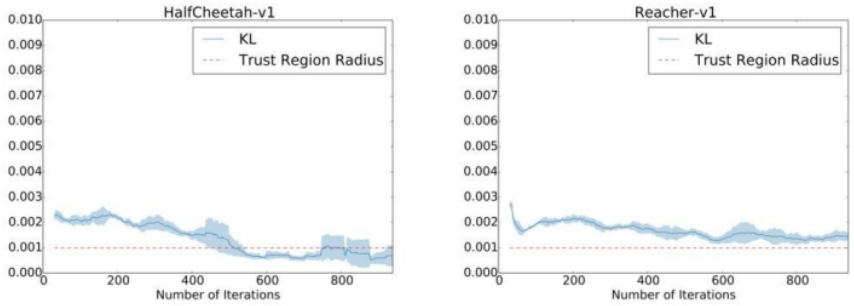


图 9：该图显示了在使用 ACKTR 进行信赖域优化训练期间的精确 KL 变化。实际 KL 接近信赖域半径，显示了通过 Kronecker 分解近似实现信赖域优化的有效性

F vec 以及克罗内克积

1. vec 矢量化算子（Vectorization）定义若矩阵 $A = (a_{ij}) \in \mathbb{R}^{m\times n}$，则 $\text{vec}(A)$ 是将 A 的列按顺序堆叠成的 $mn \times 1$ 列向量，其定义为：

$$\text{vec}(A) = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \\ a_{12} \\ a_{22} \\ \vdots \\ a_{mn} \end{bmatrix}$$

2. 克罗内克积（Kronecker Product）设 A 是 $m \times n$ 矩阵，B 是 $p \times q$ 矩阵，则克罗内克积 $A \otimes B$ 是一个 $(m \cdot p) \times (n \cdot q)$ 矩阵，定义为：

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$$

其中 a_{ij} 是矩阵 A 的第 i 行第 j 列元素，$a_{ij}B$ 表示用 a_{ij} 对 B 进行数乘（即 B 的每个元素都乘以 a_{ij}）。

克罗内克积的核心性质 (1) 结合律: $(A \otimes B) \otimes C = A \otimes (B \otimes C)$

(2) 分配律:

$$A \otimes (B + C) = A \otimes B + A \otimes C$$
$$(A + B) \otimes C = A \otimes C + B \otimes C$$

(3) 矩阵乘积性质: 若 A 与 C 可乘、B 与 D 可乘，则 $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$

(4) 转置性质: $(A \otimes B)^\top = A^\top \otimes B^\top$

(5) 逆矩阵性质: 若 A、B 均可逆，则 $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$

自然策略梯度[1,2]

摘要

我们提供了一种自然梯度方法，它基于参数空间的底层结构表示最陡下降方向。虽然梯度方法不能在参数值上做出大的改变，但我们展示了自然梯度是朝着选择贪婪最优动作的方向移动，而不仅仅是选择更好的动作。这些贪婪最优动作是在使用近似、兼容的价值函数进行策略迭代一次改进步骤下会被选择的动作，如 Sutton 等人 [9] 所定义。随后，我们展示了这种方法在简单 MDP 以及更具挑战性的俄罗斯方块 MDP 中的显著性能提升。

1 引言

目前，直接策略梯度（Policy-Gradient, PG）方法受到了广泛关注。这类方法旨在大型马尔可夫决策问题中，通过沿着未来奖励的梯度下降，从受限策略类中寻找较优策略 π。遗憾的是，标准梯度下降规则并不具备协变性。粗略地说，规则 $\Delta\theta_i = \alpha\partial J/\partial\theta_i$ 存在量纲不一致性，因为左侧的量纲与 θ_i 一致，而右侧的量纲为 $1/\theta_i$（且并非所有 θ_i 的量纲都相同）

在本文中，我们通过基于策略的底层结构定义度量来提出一个协变梯度。我们通过展示自然梯度朝着选择贪婪最优动作的方向移动，从而与策略迭代建立联系。随后，我们分析了自然梯度在简单和复杂马尔可夫决策过程（MDP）中的性能。与 Amari 的研究结果一致 [1]，我们的工作表明，使用该方法时，平台现象可能不会那么严重。

2 自然梯度

有限马尔可夫决策过程（MDP）是一个元组 (S, s_0, A, R, P)，其中：S 是有限状态集合，s_0 是初始状态，A 是有限动作集合，R 是奖励函数 $R : S \times A \to [0, R_{\max}]$，$P$ 是状态转移模型。智能体的决策过程由随机策略 $\pi(a; s)$ 刻画，该策略表示在状态 s 下选择动作 a 的概率（分号用于区分随机变量与概率分布的参数）。

我们假设所有策略 π 均满足遍历性，即存在定义良好的平稳分布 ρ^π。在此假设下，平均奖励（或无折扣奖励）定义为 $\eta(\pi) = \sum_{s,a} \rho^\pi(s)\pi(a; s)R(s, a)$，状态-动

[1]原文：Sham Kakade . A Natural Policy Gradient.

[2]译者：徐阳阳。在原文基础上增补了部分推导细节，文中图片均为原文截屏。

作值函数定义为 $Q^\pi(s,a) = \mathbb{E}_\pi\{\sum_{t=0}^{\infty} R(s_t,a_t) - \eta(\pi) \mid s_0 = s,\ a_0 = a\}$，值函数定义为 $J^\pi(s) = \mathbb{E}_{\pi(a';s)}\{Q^\pi(s,a')\}$，其中 s_t 和 a_t 分别表示时刻 t 的状态与动作。

我们考虑更具挑战性的场景：智能体的目标是在一类光滑参数化的受限策略集合 $\Pi = \{\pi^\theta \mid \theta \in \mathbb{R}^m\}$ 中，寻找使平均奖励最大化的策略，其中 π^θ 表示参数化策略 $\pi(a;s,\theta)$。平均奖励的精确梯度（参见文献 [8,9]）为：

$$\nabla\eta(\theta) = \sum_{s,a} \rho^\pi(s)\nabla\pi(a;s,\theta)Q^\pi(s,a) \tag{1}$$

此处我们简化记法，用 $\eta(\theta)$ 替代 $\eta(\pi^\theta)$。平均奖励 $\eta(\theta)$ 的最速下降方向定义为：在参数更新步长的平方长度 $|d\theta|^2$ 固定为某一小常数的约束下，使 $\eta(\theta + d\theta)$ 最小化的向量 $d\theta$。该平方长度基于正定矩阵 $G(\theta)$ 定义，即 $|d\theta|^2 = \sum_{i,j} G_{ij}(\theta)d\theta_i d\theta_j = d\theta^\top G(\theta)d\theta$（采用向量形式表示）。此时最速下降方向为 $G^{-1}\nabla\eta(\theta)$ [1]。标准梯度下降采用的方向是 $\nabla\eta(\theta)$，这等价于假设 $G(\theta)$ 为单位矩阵 I 时的最速下降方向。然而，这种人为选择的度量矩阵未必合理。正如 Amari [1] 所提出的，更优的度量矩阵不应依赖于坐标的选择，而应基于这些坐标所参数化的流形（即曲面）本身。该度量矩阵定义的梯度即为自然梯度。

尽管我们简化记法使用 $\eta(\theta)$，但平均奖励本质上是定义在策略分布集合 $\{\pi^\theta : \theta \in \mathbb{R}^m\}$ 上的函数。对每个状态 s，均对应一个概率流形，策略分布 $\pi(a;s,\theta)$ 是该流形上以 θ 为坐标的点。此分布 $\pi(a;s,\theta)$ 对应的 Fisher 信息矩阵为

$$F_s(\theta) = \mathbb{E}_{\pi(a;s,\theta)}\left[\frac{\partial \log \pi(a;s,\theta)}{\partial\theta_i} \cdot \frac{\partial \log \pi(a;s,\theta)}{\partial\theta_j}\right] \tag{2}$$

显然，该矩阵是正定的。正如 Amari（见 [1]）所示，Fisher 信息矩阵在相差一个比例因子的意义下，是概率分布参数空间上的不变度量。这里的“不变性”指：无论选择何种坐标（即参数化方式），该矩阵对两点间距离的定义始终一致——这与取 $G = I$ 作为度量的情况不同。

(Fisher 信息诱导度量的坐标不变性) 设统计模型 $\{p(x;\theta) : \theta \subset \mathbb{R}^d\}$，其 Fisher 信息矩阵定义为

$$F_{ij}(\theta) = \mathbb{E}_\theta\left[\frac{\partial \log p(x;\theta)}{\partial\theta_i}\frac{\partial \log p(x;\theta)}{\partial\theta_j}\right].$$

由此定义双线性形式

$$G_\theta(u,v) := u^\top F(\theta)\, v$$

对任意光滑可逆的重参数化

$$\theta' = \varphi(\theta),$$

以及相应的切向量变换

$$u' = \frac{\partial \theta'}{\partial \theta} u, \qquad v' = \frac{\partial \theta'}{\partial \theta} v,$$

都有

$$G_\theta(u, v) = G_{\theta'}(u', v').$$

即，Fisher 信息诱导的黎曼度量在坐标变换下保持不变。

证明： 定义

$$s_i(x;\theta) := \frac{\partial \log p(x;\theta)}{\partial \theta_i}.$$

令 $\theta' = \varphi(\theta)$，记

$$J_{ik} := \frac{\partial \theta_i}{\partial \theta'_k}.$$

由链式法则，

$$\frac{\partial \log p(x;\theta)}{\partial \theta'_k} = \sum_i \frac{\partial \theta_i}{\partial \theta'_k} \frac{\partial \log p(x;\theta)}{\partial \theta_i} = \sum_i J_{ik}\, s_i(x;\theta).$$

因此，新坐标下的 Fisher 信息矩阵为

$$\begin{aligned} F'_{kl}(\theta') &= \mathbb{E}_\theta \left[\frac{\partial \log p}{\partial \theta'_k} \frac{\partial \log p}{\partial \theta'_l} \right] \\ &= \mathbb{E}_\theta \left[\sum_{i,j} J_{ik} J_{jl} s_i s_j \right] \\ &= \sum_{i,j} J_{ik} F_{ij}(\theta) J_{jl}. \end{aligned}$$

写成矩阵形式即

$$F'(\theta') = J^\top F(\theta) J.$$

另一方面，切向量在新坐标下满足

$$u' = J^{-1} u, \qquad v' = J^{-1} v.$$

于是新坐标下的内积为

$$\begin{aligned} G_{\theta'}(u', v') &= u'^\top F'(\theta') v' \\ &= u^\top (J^{-1})^\top (J^\top F(\theta) J)(J^{-1}) v \\ &= u^\top F(\theta) v \\ &= G_\theta(u, v), \end{aligned}$$

其中使用了矩阵恒等式 $(J^{-1})^\top J^\top = I$ 与 $JJ^{-1} = I$。因此，

$$G_\theta(u, v) = G_{\theta'}(u', v'),$$

即 Fisher 信息所诱导的度量在任意光滑重参数化下保持不变。 □

由于平均奖励是定义在这一类分布上的，我们采用的直接选择为：

$$F(\theta) = \mathbb{E}_{\rho^\pi(s)}\left[F_s(\theta)\right] \tag{3}$$

其中期望是关于策略 π^θ 的平稳分布取的。注意，尽管每个 F_s 独立于马尔可夫决策过程的转移模型，但对平稳分布的期望会引入参数间的依赖关系。直观上，$F_s(\theta)$ 度量了对应于单个状态的概率流形上的距离，而 $F(\theta)$ 是这类距离的均值。此时的最速下降方向为：

$$\widetilde{\nabla}\eta(\theta) = F(\theta)^{-1}\nabla\eta(\theta) \tag{4}$$

3 方法

本节对比自然梯度下的策略改进方法与策略迭代方法。为使对比具备合理性，我们考虑如下场景：采用参数为 w 的适配函数逼近器 $f^\pi(s, a, w)$ 对状态-动作值函数 $Q^\pi(s, a)$ 进行逼近 w[9,6]。

3.1 兼容函数近似

对于向量 $\omega, \theta \in \mathbb{R}^m$，我们定义：

$$\psi(s, a)^\pi = \nabla \log \pi(a; s, \theta), \quad f^\pi(s, a; \omega) = \omega^\top \psi^\pi(s, a) \tag{5}$$

其中 $[\nabla \log \pi(a; s, \theta)]_i = \partial \log \pi(a; s, \theta)/\partial\theta_i$。令 $\widetilde{\omega}$ 最小化近似误差 $e(\omega, \pi) = \sum_{s,a} \rho^\pi(s)\pi(a; s, \theta)\left(f^\pi(s, a; \omega) - Q^\pi(s, a)\right)^2$。若在梯度计算（式 (1)）中使用该逼近器 $f^\pi(s, a; \widetilde{\omega})$ 替代真实值计算梯度，则最终结果仍为真实值（参考文献 [6]），因此这是 Actor-Critic 类算法中梯度计算的合理选择。

定理 1：设 $\widetilde{w}$ 是最小化平方误差 $e(w, \pi_\theta)$，则

$$\widetilde{\omega} = \widetilde{\nabla}\eta(\theta)$$

证明：由于 $\widetilde{w}$ 最小化平方误差，故满足条件 $\partial e/\partial w_i = 0$，这意味着：

$$\sum_{s,a} \rho^\pi(s)\pi(a; s, \theta)\psi^\pi(s, a)\left(\psi^\pi(s, a)^\top \widetilde{w} - Q^\pi(s, a)\right) = 0$$

等价于：

$$\left(\sum_{s,a}\rho^{\pi}(s)\pi(a;s,\theta)\psi^{\pi}(s,a)\psi^{\pi}(s,a)^{\top}\right)\widetilde{w}=\sum_{s,a}\rho^{\pi}(s)\pi(a;s,\theta)\psi^{\pi}(s,a)Q^{\pi}(s,a)$$

由 ψ^{π} 的定义，$\nabla\pi(a;s,\theta)=\pi(a;s,\theta)\psi^{\pi}(s,a)$，因此上式右侧等于 $\nabla\eta$。同时由 ψ^{π} 的定义，$F(\theta)=\sum_{s,a}\rho^{\pi}(s)\pi(a;s,\theta)\psi^{\pi}(s,a)\psi^{\pi}(s,a)^{\top}$。代入后可得：

$$F(\theta)\widetilde{w}=\nabla\eta(\theta)$$

求解 $\widetilde{w}$ 得 $\widetilde{w}=F(\theta)^{-1}\nabla\eta(\theta)$，结合自然梯度的定义即可得证。 □

因此，合理的 Actor-Critic 框架（即使用 $f^{\pi}(s,a;w)$ 的框架）必须将自然梯度用作线性函数逼近器的权重。若函数逼近器足够精确，则状态-动作值较大的动作（即具有良好状态-动作值的动作）对应的特征向量将生成较好的策略。

3.2 贪心策略改进

采用本文函数逼近器的贪婪策略改进步骤为：在状态 s 下选择动作 a，当且仅当 $a\in\arg\max_{a'}f^{\pi}(s,a';\widetilde{w})$。本节将证明，自然梯度方向不仅会导向某一较优动作，更会趋向于这一最优动作。

首先考虑指数族策略，其形式为 $\pi(a;s,\theta)\propto\exp(\theta^{\top}\phi_{sa})$，其中 ϕ_{sa} 是 $\mathbb{R}^m$ 中的某一特征向量。选择指数族策略的原因在于其具有仿射几何结构（即平面的平坦几何特性），因此沿切向量平移流形上的点后，该点仍会处于流形上。一般而言，粗略地说，策略 $\pi(a;s,\theta)$ 对应的概率流形可能是弯曲的，此时沿切向量平移流形上的点未必能使该点保持在流形上（例如球面的情况）。后续我们将讨论一般（非指数族）情形。

我们现在展示，对于指数族，一个在自然梯度方向上足够大的步长将导致一个策略，该策略等同于在贪婪策略改进步骤后得到的策略。

定理 2. 对策略 $\pi(a;s,\theta)\propto\exp(\theta^{\top}\phi_{sa})$，假设 $\widetilde{\nabla}\eta(\theta)\neq 0$，且 $\widetilde{w}$ 最小化近似误差。令 $\pi_{\infty}(a;s)=\lim_{\alpha\to\infty}\pi(a;s,\theta+\alpha\widetilde{\nabla}\eta(\theta))$，则 $\pi_{\infty}(a;s)\neq 0$ 当且仅当 $a\in\arg\max_{a'}f^{\pi}(s,a';\widetilde{w})$。

证明： 由前文结论，$f^{\pi}(s,a;\widetilde{w})=\widetilde{\nabla}\eta(\theta)^{\top}\psi^{\pi}(s,a)$。根据 $\pi(a;s,\theta)$ 的定义，$\psi^{\pi}(s,a)=\phi_{sa}-\mathbb{E}_{\pi(a';s,\theta)}(\phi_{sa'})$。由于 $\mathbb{E}_{\pi(a';s,\theta)}(\phi_{sa'})$ 与 a 无关，可得：

$$\arg\max_{a'}f^{\pi}(s,a';\widetilde{w})=\arg\max_{a'}\widetilde{\nabla}\eta(\theta)^{\top}\phi_{sa'}$$

$$
\begin{aligned}
\psi^{\pi}(s,a) &= \nabla_\theta \log \pi(a;s,\theta) \\
&= \nabla_\theta \left(\theta^T \phi_{sa}\right) - \nabla_\theta \log Z(s,\theta) \\
&= \phi_{sa} - \frac{1}{Z(s,\theta)} \nabla_\theta Z(s,\theta) \\
&= \phi_{sa} - \frac{1}{Z(s,\theta)} \sum_{a'} \phi_{sa'} \exp(\theta^T \phi_{sa'}) \\
&= \phi_{sa} - \sum_{a'} \phi_{sa'} \pi(a';s,\theta) \\
&= \phi_{sa} - \mathbb{E}_{\pi(a';s,\theta)}[\phi_{sa'}]
\end{aligned}
$$

执行梯度步后，$\pi_\alpha(a;s,\theta+\widetilde{\alpha}\nabla\eta(\theta)) \propto \exp(\theta^\top\phi_{sa}+\alpha\widetilde{\nabla}\eta(\theta)^\top\phi_{sa})$。由于 $\widetilde{\nabla}\eta(\theta)\neq 0$，显然当 $\alpha\to\infty$ 时，项 $\alpha\widetilde{\nabla}\eta(\theta)^\top\phi_{sa}$ 起主导作用，因此 $\pi_\infty(a;s)=0$ 当且仅当 $a\notin \arg\max_{a'}\widetilde{\nabla}\eta(\theta)^\top\phi_{sa'}$。

$$
\exp\big((\theta+\alpha\bar{\nabla}\eta)^\top\phi_{sa}\big) = \exp(\theta^\top\phi_{sa})\exp(\alpha\bar{\nabla}\eta^\top\phi_{sa}).
$$

因此

$$
\pi_\alpha(a;s,\theta) = \frac{\exp(\theta^\top\phi_{sa})\exp(\alpha\bar{\nabla}\eta^\top\phi_{sa})}{\sum_{a'\in\mathcal{A}}\exp(\theta^\top\phi_{sa'})\exp(\alpha\bar{\nabla}\eta^\top\phi_{sa'})}.
$$

令

$$
M := \max_{a\in\mathcal{A}} \bar{\nabla}\eta^\top\phi_{sa},
$$

并对分子与分母同时除以 $\exp(\alpha M)$，得

$$
\pi_\alpha(a;s,\theta) = \frac{\exp(\theta^\top\phi_{sa})\exp\big(\alpha(\bar{\nabla}\eta^\top\phi_{sa}-M)\big)}{\sum_{a'\in\mathcal{A}}\exp(\theta^\top\phi_{sa'})\exp\big(\alpha(\bar{\nabla}\eta^\top\phi_{sa'}-M)\big)}.
$$

若 $a\notin\mathcal{A}^\star$，则

$$
\bar{\nabla}\eta^\top\phi_{sa} - M < 0,
$$

从而

$$
\lim_{\alpha\to\infty}\exp\big(\alpha(\bar{\nabla}\eta^\top\phi_{sa}-M)\big)=0.
$$

正是在这个意义上，自然梯度趋向于导向“选择最优动作”的方向。可以直接证明：若使用标准的非协变梯度规则，则策略仅会选择“较优动作”（不一定是最优动作），即它会选择使 $f^\pi(s,a;\widetilde{w}) > \mathbb{E}_{\pi(a';s,\theta)}(f^\pi(s,a';\widetilde{w}))$ 的动作。指数族策略是这类情况的极端示例，会导致学习率趋于无穷大。

接下来回到一般参数化策略的情形。以下定理表明：自然梯度在局部会趋向

于由局部线性逼近器 $Q^\pi(s,a)$ 所确定的最优动作。

定理 3. 假设 $\widetilde{w}$ 最小化近似误差，且参数更新为 $\theta' = \theta + \alpha\widetilde{\nabla}\eta(\theta)$，则：

$$\pi(a;s,\theta') = \pi(a;s,\theta)\left(1 + f^\pi(s,a;\widetilde{w})\right) + O(\alpha^2)$$

证明：参数 θ 的变化量 $\Delta\theta$ 为 $\alpha\widetilde{\nabla}\eta(\theta)$ ，由定理 1 可知 $\Delta\theta = \alpha\widetilde{w}$。在一阶近似下：

$$\begin{aligned}
\pi(a;s,\theta') &= \pi(a;s,\theta) + \frac{\partial\pi(a;s,\theta)}{\partial\theta}^\top \Delta\theta + O(\Delta\theta^2) \\
&= \pi(a;s,\theta)\left(1 + \psi(s,a)^\top\Delta\theta\right) + O(\Delta\theta^2) \\
&= \pi(a;s,\theta)\left(1 + \alpha\psi(s,a)^\top\widetilde{w}\right) + O(\alpha^2) \\
&= \pi(a;s,\theta)\left(1 + \alpha f^\pi(s,a;\widetilde{w})\right) + O(\alpha^2)
\end{aligned}$$

其中利用了 ψ 与 f 的定义。 □

有趣的是，值得注意的是，选择贪婪动作通常不会改善策略，许多详细的研究都致力于理解这种失败 [3]。然而，通过增加一条线搜索的开销，我们可以保证改进，并向这个贪婪的一步改进前进。由于 F 是正定的，初始改进是有保证的

4 度量与曲率

显然，我们选择的 F 并不是唯一的，问题随之而来，即是否有比 F 更好的度量可用。在参数估计的不同情况下，Fisher 信息会收敛到 Hessian，因此它在渐近意义上是有效的 [1]，即达到 Cramer-Rao 界限。我们的情况更类似于盲源分离的情形，在这种情况下，度量是基于底层参数空间 [1]（非奇异矩阵）选择的，并不一定在渐近意义上有效（即不达到二阶收敛）。正如 Mackay [7] 所论述的，一种策略是从 Hessian 的数据无关项中提取度量（如果可能的话），事实上，Mackay [7] 在盲源分离的情况下得出了与 Amari 相同的结果。

尽管前文已论证我们所做选择的合理性，但我们仍希望理解 F 与海森矩阵 $\nabla^2\eta(\theta)$ 的关系——如文献 [5] 所示，该海森矩阵的形式为：

$$\nabla^2\eta(\theta) = \sum_{s,a}\rho^\pi(s)\left(\nabla^2\pi(a;s,\theta)Q^\pi(s,a) + \nabla\pi(a;s,\theta)\nabla Q^\pi(s,a)^\top + \nabla Q^\pi(s,a)\nabla\pi(a;s)^\top\right) \tag{6}$$

遗憾的是，该海森矩阵中的所有项均依赖于数据（即与状态-动作值耦合）。显然，由于最后两项包含 ∇Q^π，F 并未捕捉这些项的任何信息。不过，第一项可能与 F 存在关联, 这源于其中的 $\nabla^2\pi$ 因子。但 Q 值会对我们策略的曲率进行加权，而我们的度量矩阵却忽略了这一加权作用。

与盲源分离问题的情形类似，我们采用的度量矩阵显然未必收敛于海森矩阵，

因此也未必具备渐近效率（即无法达到二阶收敛速率）。但一般而言，海森矩阵并非正定矩阵，其提供的曲率信息在参数 θ 接近局部极大值前实用价值有限。而共轭梯度类方法在局部极大值附近的效率预计会更高。

5 实验

我们首先在几个简单的马尔可夫决策过程（MDP）中验证自然梯度的性能，随后在更具挑战性的俄罗斯方块（Tetris）马尔可夫决策过程中展开分析。由于估计平均奖励梯度 $\nabla\eta(\theta)$ 时必须计算对数策略梯度 $\nabla \log \pi$，因此可通过在线方式直接估计费希尔信息矩阵 F。在长度为 T 的轨迹中，采用如下更新规则：

$$f \leftarrow f + \nabla \log \pi(a_t; s_t, \theta) \nabla \log \pi(a_t; s_t, \theta)^\top$$

此时 f/T 是 F 的一致估计。在前两个算例中，我们暂不考虑采样问题，而是对精确梯度进行数值积分（参数更新形式为 $\theta_t = \theta_0 + \int_0^t \nabla\eta(\theta_t)dt$）。在所有仿真中，策略往往会收敛到确定性策略（此时 $\nabla \log \pi \to 0$）；为避免矩阵 F 奇异，我们在每一步更新中均添加约 $10^{-3}I$（I 为单位矩阵）。

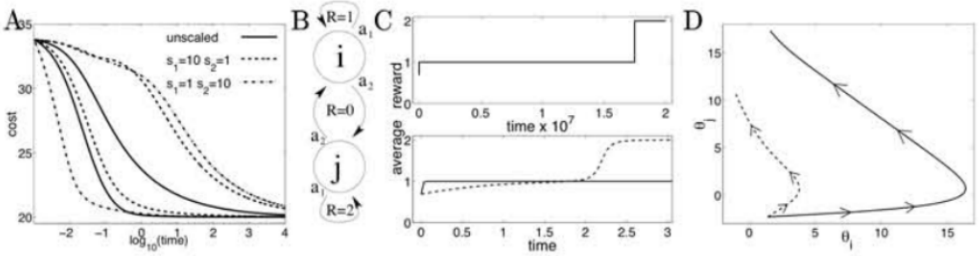
我们在一个简单的一维线性二次调节器（linear quadratic regulator, LQR）中对自然策略梯度进行仿真。该系统的动力学方程为：

$$x(t+1) = 0.7x(t) + u(t) + \xi(t)$$

其中噪声项 $\xi \sim \mathcal{N}(0, 1)$（服从均值为 0、方差为 1 的高斯分布）。控制目标为施加控制信号 u 使系统状态维持在 $x = 0$（每步产生的代价为 $x(t)^2$）。采用的参数化策略为：

$$\pi(u; x, \theta) \propto \exp(\theta_1 x^2 + \theta_2 x)$$

图 1A 展示了当参数的量纲被放大 10 倍时的性能提升效果（详见图注）。注意，获得约 22 分性能指标的时间相差约三个数量级。收敛速度快了约三个数量级。另需注意，不同缩放比例下的曲线并不完全相同。这是因为受 p_S 加权的影响，F 并非严格的不变度量。



图一：A）代价函数与 $\log_{10}$(时间) 的关系（针对 LQG，采用 20 步时间轨迹）。所使用的策略为 $\pi(u;x,\theta) \propto \exp(\theta_1 s_1 x_2 + \theta_2 s_2 x)$，其中缩放常数 s_1 和 s_2 见图例。在等效初始分布（$\theta_1 s_1 = \theta_2 s_2 = -.8$）下，最右侧三条曲线由标准梯度法生成，其余曲线由自然梯度法生成。B）详见正文。C（上）：采用 S 型策略参数化（$\pi(i;s,\theta_i) \propto \exp(\theta_i)/(1+\exp(\theta_i))$）的标准梯度下降法下，策略的平均奖励随时间的变化（时间轴为 10^7 尺度），初始条件为 $\pi(i,1) = .8$ 且 $\pi(j,1) = .1$。C（下）：上述图表早期窗口内，标准梯度下降法（实线）和自然梯度下降法（虚线）下平均奖励随时间的变化（时间轴无缩放）。D）标准梯度法（实线）和自然梯度法（虚线）的相空间图。

在一个简单的两状态马尔可夫决策过程（图 1B）中，$p(s)$ 的加权效应尤为明显——该过程包含自转移和交叉转移动作及奖励（详见图示）。增加状态 i 的自环概率会降低状态 j 的平稳分布概率。采用 sigmoid 参数化策略（见图注），并设置初始条件为 $p(i) = 0.8$、$p(j) = 0.2$ 时，梯度规则会初始提升两个状态的自环动作概率（因为单步策略改进会为每个状态选择自环动作）。由于标准梯度通过 $p(s)$ 对各参数的学习过程进行加权（见式 1），状态 i 的自环动作概率提升速度快于状态 j，这进一步降低了状态 j 的有效学习率，导致平均奖励陷入值为 1 的平坦平台期（如图 1C 上半部分所示），此时状态 j 的学习因低平稳概率而受阻。

该问题极为严重：在达到最优策略前，$p(j)$ 从初始值 0.2 降至 10^{-7}，这对采样方法而言是灾难性的。图 1C 下半部分展示了自然梯度在图 1C 上半部分早期时间窗口的性能——不仅达到最优策略的时间缩短了 10^7 倍，状态 i 的平稳分布概率也从未低于 0.05。需注意，标准梯度虽在初始阶段提升平均奖励的速度更快，但最终会陷入状态 i 的局部最优。图 1D 的相空间图显示了不同参数学习过程的不均衡性，这正是问题的核心。

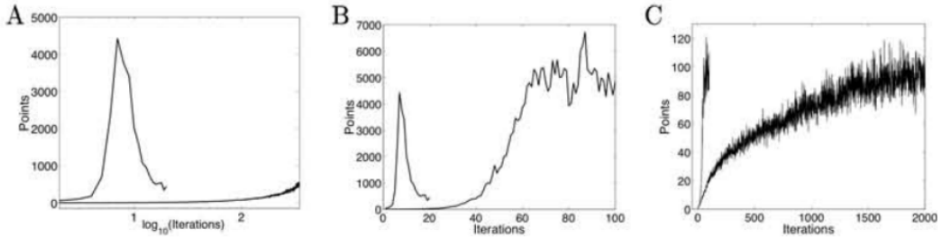


图 2：A）得分与 $\log_{10}$(迭代次数) 的关系。最上方曲线采用与文献 [3] 相同的特征（该特征为游戏中各列高度与空洞数量的简单函数），复现了文献 [3] 的实验结果。对于该性能退化现象，我们与文献 [3] 均未给出合理解释。下方曲线则体现了标准梯度法的较差性能表现。B）右侧曲线对应自然策略梯度法的实验结果（其中采用了文献 [2] 提出的有偏梯度法，需注意该方法单独使用时性能表现较差）。我

们发现，若对添加到矩阵 F 中的 $10^{-3}I$ 鲁棒化因子进行更精细的参数控制，模型的性能提升速度与收敛上限均可得到优化（注：本实验未对相关参数进行精细调节）。C）由于这类仿真实验对计算资源要求较高，我们在规模更小的俄罗斯方块游戏中开展梯度法实验（游戏中列高度设为 10，而非原实验的 20），验证了标准梯度法的更新曲线（右侧曲线）最终能够达到与自然梯度法（左侧曲线）相当的性能水平。

一般而言，若采用查表式玻尔兹曼策略（即 $\pi(a;s,\theta)\propto\exp(\theta_{sa})$），可直接证明自然梯度会对 $\nabla\eta$ 的各分量进行均匀加权（而非依赖 $p(s)$），从而平衡所有参数的学习过程。而梯度方法能够保证策略性能不会退化，因此俄罗斯方块成为测试这类方法的理想场景。我们采用与文献 [3] 中线性函数逼近器适配的策略（即 $\pi(a;s,\theta)\propto\exp(\theta^{\top}\phi_{sa})$，其中 ϕ_{sa} 为相同的特征向量）。文献 [3] 所用特征包括：每列的高度、相邻列的高度差、最大高度以及“空洞”数量。图 2A 的下曲线显示标准梯度方法的性能表现极差。为加快学习速度，我们尝试了多种更复杂的改进方法，但均未取得效果，包括共轭梯度法、权重衰减、学习率退火、文献 [2] 中的方差减少方法、式 (6) 中的海森矩阵方法等。图 2B 展示了自然梯度带来的显著性能提升（注意此时时间轴为线性刻度）。这一结果与第 3 节的理论分析一致——自然梯度的更新方向趋向于贪婪策略改进的解。尽管自然梯度的收敛速度略慢于贪婪策略迭代（图 2B 左曲线），但这是小步长更新下的预期结果，且梯度方法不会导致策略性能退化。图 2 显示，在该游戏的简化版本中（详见图注），标准梯度规则（右曲线）的性能最终会达到与自然梯度相当的水平，只是整体进程呈比例放缓。

6 讨论

尽管与贪婪策略迭代相比，梯度方法不能进行大的策略变更，但第 3 节暗示这两种方法可能并没有那么不同，因为自然梯度方法正在朝着策略改进步骤的解移动。加上线搜索的开销，这两种方法甚至更为相似。其优势在于性能改进现在是有保证的，这与贪婪策略迭代步骤不同。有趣且遗憾的是，F 并不会渐近收敛到 Hessian，因此共轭梯度方法在渐近意义上可能更合理。然而，在远离收敛点的情况下，Hessian 并不一定提供有用信息，而自然梯度可能更有效（如在俄罗斯方块中所示）。自然梯度在远离最大值的情况下仍能高效的直觉是，它正在推动策略选择贪婪的最优动作。通常，在参数空间中远离最大值的区域是性能变化可能较大的地方。充分接近最大值时，由于梯度较小，性能变化很小，因此尽管共轭方法在接近最大值时可能收敛更快，但相应的性能变化可能微不足道。还需要更多的实验工作来进一步理解自然梯度的有效性。

致谢

我们感谢 Emo Todorov 和 Peter Dayan 的许多有益讨论。资金来自国家科学基金会（NSF）和加茨比慈善基金会。

参考文献

1. S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.
2. J. Baxter and P. Bartlett. Direct gradient-based reinforcement learning. Technical report, Australian National University, Research School of Information Sciences and Engineering, July 1999.
3. D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
4. P. Dayan and G. Hinton. Using em for reinforcement learning. *Neural Computation*, 9:271–278, 1997.
5. S. Kakade. Optimizing average reward using discounted reward. *COLT*, in press., 2001.
6. V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12, 2000.
7. D. MacKay. Maximum likelihood and covariant algorithms for independent component analysis. Technical report, University of Cambridge, 1996.
8. P. Marbach and J. Tsitsiklis. Simulation-based optimization of markov reward processes. Technical report, Massachusetts Institute of Technology, 1998.
9. R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Neural Information Processing Systems*, 13, 2000.
10. L. Xu and M. I. Jordan. On convergence properties of the EM algorithm for gaussian mixtures. *Neural Computation*, 8(1):129–151, 1996.