具身智能大作业报告

姓　　名　卢祥云

学　　号　1120230944

邮　　箱　1435778533@qq.com

北京理工大学

2026 年 2 月 14 日

目录

第一章　基本原理

1.1　强化学习基础

1.1.1　马尔可夫决策过程

强化学习问题通常被形式化为马尔可夫决策过程 (Markov Decision Process, MDP)，定义为五元组 (S, A, P, R, γ)：S 为状态空间，A 为动作空间，$P(s'|s,a)$ 为状态转移概率，$R(s,a,s')$ 为奖励函数，$\gamma \in [0,1]$ 为折扣因子。在每个时间步 t，智能体处于状态 $s_t \in S$，根据策略 π 选择动作 $a_t \in A$，环境返回奖励 $r_t = R(s_t, a_t, s_{t+1})$ 并转移到新状态 $s_{t+1} \sim P(\cdot|s_t, a_t)$。

1.1.2　回报与价值函数

从时刻 t 开始的**累积折扣回报**定义为 $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$。

定义 1.1 (状态价值函数). 从状态 s 出发，遵循策略 π 的期望回报：$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$。

定义 1.2 (动作价值函数). 在状态 s 采取动作 a，之后遵循策略 π 的期望回报：$Q^\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$。

两者的关系为 $V^\pi(s) = \sum_{a \in A} \pi(a|s)\, Q^\pi(s,a)$。

1.1.3　Bellman 方程

价值函数满足递归关系，即 **Bellman 方程**：

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V^\pi(s') \right] \tag{1.1}$$

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a') \right] \tag{1.2}$$

最优策略 π^* 对应的 **Bellman 最优方程**为：

$$Q^*(s,a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q^*(s',a') \,\middle|\, s, a \right] \tag{1.3}$$

1.2　DQN (2013)：深度 Q 网络

1.2.1　算法背景

2013 年，Mnih 等人在论文 *"Playing Atari with Deep Reinforcement Learning"* 中首次提出深度 Q 网络 (Deep Q-Network, DQN)，成功地将深度神经网络与强化学习相结合，直接从高维原始像素输入学习控制策略，是深度强化学习领域的奠基性工作。传统 Q-Learning 使用表格存储每个状态-动作对的 Q 值，但在状态空间巨大（如 Atari 游戏的像素空间）时表格方法不可行。DQN 的核心思想是使用深度神经网络 $Q(s,a;\theta)$ 作为函数逼近器来近似最优动作价值函数 $Q^*(s,a)$。

1.2.2　Q-Learning 基础

Q-Learning 是一种无模型的离策略时序差分 (TD) 学习算法，其更新规则为：

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1.4}$$

其中 α 为学习率，$r + \gamma \max_{a'} Q(s',a')$ 为 TD 目标，$\delta = r + \gamma \max_{a'} Q(s',a') - Q(s,a)$ 为 TD 误差。

1.2.3　核心创新：经验回放

DQN 2013 的关键创新是**经验回放** (Experience Replay) 机制。智能体将交互经验 (s_t, a_t, r_t, s_{t+1}) 存储在回放缓冲区 $\mathcal{D}$ 中，训练时从中均匀随机采样 mini-batch：$(s_j, a_j, r_j, s_{j+1}) \sim \mathrm{Uniform}(\mathcal{D})$。经验回放的作用有三：(1) **打破时序相关性**，随机采样消除了连续样本间的相关性，满足随机梯度下降对 i.i.d. 样本的要求；(2) **提高数据利用效率**，每条经验可被多次采样用于训练；(3) **平滑数据分布**，缓冲区中混合了不同时期的经验，减缓了数据分布的剧烈变化。

1.2.4　状态预处理

对于 Atari 游戏，原始帧 ($210 \times 160 \times 3$) 依次经过灰度化、下采样至 84×84、堆叠最近 4 帧构成状态 $\phi_t = (x_{t-3}, x_{t-2}, x_{t-1}, x_t) \in \mathbb{R}^{4\times84\times84}$，以捕获运动信息。此外对奖励进行裁剪 $r_{\text{clipped}} = \mathrm{sign}(r)$，统一不同游戏的奖励尺度。

1.2.5　网络结构

DQN 2013 使用卷积神经网络直接从像素输入提取特征，网络结构如表 1.1 所示。网络输入为 $\phi(s) \in \mathbb{R}^{4\times84\times84}$，输出为所有动作对应的 Q 值向量 $Q(\phi(s), \cdot\,; \theta) \in \mathbb{R}^{|A|}$。

表 1.1： DQN 2013 网络结构

层	输入尺寸	输出尺寸	配置	激活
Conv1	$4 \times 84 \times 84$	$16 \times 20 \times 20$	16 个 8×8 核, stride=4	ReLU
Conv2	$16 \times 20 \times 20$	$32 \times 9 \times 9$	32 个 4×4 核, stride=2	ReLU
Flatten	$32 \times 9 \times 9$	2592	—	—
FC	2592	256	全连接	ReLU
Output	256	$\lvert A \rvert$	全连接	线性

1.2.6　损失函数与优化

对于从回放缓冲区采样的 mini-batch，定义均方误差损失：

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(y - Q(s,a;\theta) \right)^2 \right], \quad y = \begin{cases} r & \text{if terminal} \\ r + \gamma \max_{a'} Q(s',a';\theta) & \text{otherwise} \end{cases} \tag{1.5}$$

梯度为 $\nabla_\theta L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(y - Q(s,a;\theta)) \nabla_\theta Q(s,a;\theta)]$。计算 TD 目标 y 时使用 stop-gradient 操作，即 y 不参与反向传播。本实现使用 RMSProp 优化器。

1.2.7　探索策略

DQN 采用 ε-greedy 策略平衡探索与利用：以概率 ε 选择随机动作，否则选择 $\arg\max_a Q(\phi(s_t), a; \theta)$。$\varepsilon$ 从 1.0 线性衰减至 0.1：

$$\varepsilon(t) = \begin{cases} \varepsilon_{\text{start}} - \dfrac{t}{T}(\varepsilon_{\text{start}} - \varepsilon_{\text{end}}) & \text{if } t < T \\ \varepsilon_{\text{end}} & \text{otherwise} \end{cases} \tag{1.6}$$

其中 $T = 1{,}000{,}000$ 为衰减帧数。

1.2.8　完整算法

DQN 2013 的完整算法如算法 1 所示。

Algorithm 1 DQN 2013 算法

初始化回放缓冲区 $\mathcal{D}$（容量 N），初始化 Q 网络参数 θ
for episode $= 1$ to M **do**
　初始化状态 s_1，预处理得到 $\phi_1 = \phi(s_1)$
　for $t = 1$ to T **do**
　　以概率 ε 选择随机动作 a_t，否则 $a_t = \arg\max_a Q(\phi_t, a; \theta)$
　　执行 a_t，观察 r_t, s_{t+1}，将 $(\phi_t, a_t, r_t, \phi_{t+1})$ 存入 $\mathcal{D}$
　　采样 mini-batch，计算 $y_j = r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta)$（终止时 $y_j = r_j$）
　　对 $(y_j - Q(\phi_j, a_j; \theta))^2$ 执行梯度下降更新 θ
　end for
end for

1.3　DQN (2015)：Nature DQN 改进版

1.3.1　DQN 2013 的训练不稳定问题

DQN 2013 存在根本性的训练不稳定问题：同一个网络同时用于计算当前 Q 值 $Q(s, a; \theta)$ 和 TD 目标 $y = r + \gamma \max_{a'} Q(s', a'; \theta)$。每次更新 θ 后 TD 目标也随之改变，造成“追逐移动目标”的困境，导致训练震荡甚至发散。

1.3.2　核心改进：目标网络

2015 年，Mnih 等人在 *Nature* 发表 *“Human-level control through deep reinforcement learning”*，引入**目标网络** (Target Network) 解决上述问题。Nature DQN 维护两个结构相同但参数不同的网络：主网络（参数 θ，用于选择动作和梯度更新）与目标网络（参数 θ^-，仅用于计算 TD 目标，不参与梯度更新）。TD 目标改为：

$$y = \begin{cases} r & \text{if terminal} \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{otherwise} \end{cases} \tag{1.7}$$

目标网络参数每隔 C 步（本实现 $C = 10{,}000$）从主网络硬拷贝：$\theta^- \leftarrow \theta$。在两次同步之间 TD 目标保持稳定，显著提升训练稳定性。

1.3.3　网络结构改进

Nature DQN 使用了更深更宽的网络，对比如表 1.2 所示。特征图尺寸变化：$4 \times 84 \times 84 \xrightarrow{\text{Conv1}} 32 \times 20 \times 20 \xrightarrow{\text{Conv2}} 64 \times 9 \times 9 \xrightarrow{\text{Conv3}} 64 \times 7 \times 7 \xrightarrow{\text{Flatten}} 3136 \xrightarrow{\text{FC}} 512 \to |A|$。

表 1.2: DQN 2013 与 DQN 2015 网络结构对比

层	DQN 2013	DQN 2015 (Nature)	变化
Conv1	16 个 8×8, stride=4	**32** 个 8×8, stride=4	滤波器翻倍
Conv2	32 个 4×4, stride=2	**64** 个 4×4, stride=2	滤波器翻倍
Conv3	—	**64** 个 3×3, stride=1	**新增**
FC	256	**512**	隐藏单元翻倍
参数量	∼1.7M	∼3.5M	约 2 倍

1.3.4　Huber 损失函数

DQN 2015 将损失函数从 MSE 改为 **Huber 损失** (Smooth L1 Loss)，其中 $\delta = y - Q(s,a;\theta)$：

$$L_{\text{Huber}}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{if } |\delta| \le 1 \\ |\delta| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{1.8}$$

小误差区域为二次函数，保持快速收敛；大误差区域为线性函数，梯度恒为 ± 1，有效防止梯度爆炸。此外还引入梯度裁剪（最大梯度范数 10.0）进一步稳定训练。

1.3.5　与 DQN 2013 的完整对比

表 1.3 总结了两个版本的关键差异。

表 1.3: DQN 2013 与 DQN 2015 关键差异

特性	**DQN 2013**	**DQN 2015 (Nature)**
目标网络	无	有，每 C 步同步
TD 目标计算	$Q(s',a';\theta)$	$Q(s',a';\theta^-)$
网络深度	2 层卷积	3 层卷积
全连接层宽度	256	512
损失函数	MSE	Huber Loss
梯度裁剪	无	有 (max norm = 10.0)
训练稳定性	较差	显著提升

1.3.6　完整算法

Nature DQN 的完整算法如算法 2 所示。

Algorithm 2 Nature DQN 2015 算法

初始化回放缓冲区 $\mathcal{D}$（容量 N），初始化 Q-Network 参数 θ，Target Network $\theta^- = \theta$
for episode $= 1$ to M **do**
　初始化状态 s_1，预处理得到 $\phi_1 = \phi(s_1)$
　for $t = 1$ to T **do**
　　以概率 ε 选择随机动作 a_t，否则 $a_t = \arg\max_a Q(\phi_t, a; \theta)$
　　执行 a_t，观察 r_t, s_{t+1}，将 $(\phi_t, a_t, r_t, \phi_{t+1})$ 存入 $\mathcal{D}$
　　采样 mini-batch，用**目标网络**计算 $y_j = r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta^-)$（终止时 $y_j = r_j$）
　　对 $L_{\text{Huber}}(y_j - Q(\phi_j, a_j; \theta))$ 执行梯度下降更新 θ
　　每 C 步同步：$\theta^- \leftarrow \theta$
　end for
end for

1.4　TRPO：信赖域策略优化

1.4.1　算法背景

Trust Region Policy Optimization (TRPO) 由 Schulman 等人于 2015 年提出。与 DQN 系列的**基于价值**方法不同，TRPO 是**基于策略**的方法，直接优化参数化策略 $\pi_\theta(a|s)$。策略梯度方法的核心挑战在于步长选择：步长过大可能导致策略性能急剧下降，步长过小则学习效率低下。TRPO 通过引入**信赖域**约束，在保证策略单调改进的同时允许尽可能大的更新步长。

1.4.2　策略梯度基础

策略梯度方法的优化目标为最大化期望累积回报 $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty}\gamma^t r_t]$。根据策略梯度定理：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\, A^{\pi_\theta}(s,a)\right] \tag{1.9}$$

其中 $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ 为**优势函数**，衡量动作 a 相对于平均水平的好坏程度。

1.4.3　策略改进的理论保证

TRPO 的理论基础来自 Kakade & Langford (2002) 的策略改进下界。新策略 $\tilde{\pi}$ 相对于旧策略 π 的性能差异为 $J(\tilde{\pi}) = J(\pi) + \mathbb{E}_{s\sim d^{\tilde{\pi}},a\sim\tilde{\pi}}[A^\pi(s,a)]$，其中 $d^{\tilde{\pi}}$ 为折扣状态访问频率。由于 $d^{\tilde{\pi}}$ 依赖未知新策略，TRPO 构造替代目标，并使用重要性采样改写为：

$$L_{\theta_{\text{old}}}(\theta) = \mathbb{E}_{s\sim d^{\pi_{\text{old}}},a\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\text{old}}}(s,a)\right] \tag{1.10}$$

1.4.4　信赖域约束

仅最大化替代目标不能保证实际性能提升。Schulman 等人证明了策略改进下界：

定理 1.1 (TRPO 策略改进下界)**.** 令 $D_{KL}^{\max}(\pi_{old},\pi_{new}) = \max_s D_{KL}(\pi_{old}(\cdot|s)\|\pi_{new}(\cdot|s))$，则：

$$J(\pi_{new}) \geq L_{\pi_{old}}(\pi_{new}) - \frac{4\varepsilon\gamma}{(1-\gamma)^2} D_{KL}^{\max}(\pi_{old},\pi_{new}) \tag{1.11}$$

其中 $\varepsilon = \max_{s,a}|A^\pi(s,a)|$。

该定理表明只要新旧策略间 KL 散度足够小，替代目标的改进就能保证实际性能改进。实践中 TRPO 用平均 KL 散度约束替代最大 KL 散度，得到优化问题：

$$\max_\theta \; L_{\theta_{\text{old}}}(\theta) \qquad \text{s.t.} \quad \bar{D}_{\text{KL}}(\theta_{\text{old}},\theta) \leq \delta \tag{1.12}$$

其中 δ 为信赖域半径（本实现 $\delta = 0.02$），$\bar{D}_{\text{KL}}(\theta_{\text{old}},\theta) = \mathbb{E}_{s\sim d^{\pi_{\text{old}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s)\|\pi_\theta(\cdot|s))]$。

1.4.5　自然梯度与共轭梯度法

对式 (1.12) 进行泰勒展开近似：目标函数一阶近似为 $L_{\theta_{\text{old}}}(\theta) \approx \boldsymbol{g}^T(\theta-\theta_{\text{old}})$，KL 约束二阶近似为 $\bar{D}_{\text{KL}} \approx \frac{1}{2}(\theta-\theta_{\text{old}})^T\boldsymbol{F}(\theta-\theta_{\text{old}})$，其中 $\boldsymbol{g}$ 为策略梯度，$\boldsymbol{F}$ 为 Fisher 信息矩阵。由此得到**自然梯度**更新：

$$\Delta\theta = \sqrt{\frac{2\delta}{\boldsymbol{g}^T\boldsymbol{F}^{-1}\boldsymbol{g}}}\,\boldsymbol{F}^{-1}\boldsymbol{g} \tag{1.13}$$

直接计算 $\boldsymbol{F}^{-1}$ 代价过高，TRPO 使用**共轭梯度法** (CG) 迭代求解 $\boldsymbol{F}\boldsymbol{x} = \boldsymbol{g}$，只需计算 Fisher-向量积 $\boldsymbol{F}\boldsymbol{v} = \nabla_\theta[(\nabla_\theta\bar{D}_{\text{KL}})^T\boldsymbol{v}]$（通过自动微分高效实现），避免显式构建 Fisher 矩阵。本实现使用 15 次 CG 迭代，阻尼系数 $\lambda_{\text{damp}} = 0.05$（即求解 $(\boldsymbol{F}+\lambda_{\text{damp}}\boldsymbol{I})\boldsymbol{x} = \boldsymbol{g}$）。

1.4.6　线性搜索

CG 给出的更新方向基于二阶近似，可能不够精确。TRPO 在更新方向上执行**回溯线性搜索**，确保替代目标改善且 KL 约束满足。从完整步长开始依次尝试 $\alpha\cdot\beta^k$（本实现 $\beta = 0.6$，最多 15 次），更新 $\theta_{\text{new}} = \theta_{\text{old}} + \alpha\cdot\beta^k\cdot\Delta\theta$。

1.4.7　广义优势估计 (GAE)

TRPO 使用**广义优势估计** (GAE) 计算优势函数。定义 TD 残差 $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$，则 GAE 为：

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{1.14}$$

参数 $\lambda \in [0,1]$ 控制偏差-方差权衡：$\lambda = 0$ 为单步 TD（高偏差低方差），$\lambda = 1$ 为蒙特卡洛（低偏差高方差）。本实现取 $\lambda = 0.98$，偏向考虑更长期回报。

1.4.8　Actor-Critic 架构与熵正则化

TRPO 采用 **Actor-Critic** 架构。**Actor 网络**输出动作概率分布 $\pi_\theta(a|s)$，使用 3 层卷积加全连接层，输出层为 log-softmax，采用正交初始化。**Critic 网络**估计状态价值 $V_\phi(s)$，具有相同的卷积骨架，输出单标量，通过最小化 $L_{\mathrm{critic}}(\phi) = \mathbb{E}[(V_\phi(s) - V_{\mathrm{target}})^2]$ 训练。

为鼓励探索，在策略目标中加入**熵正则化**：$L_{\mathrm{total}} = L_{\mathrm{surrogate}} + c_{\mathrm{ent}} \cdot H(\pi_\theta)$，其中 $H(\pi_\theta) = -\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)$。本实现采用衰减策略：$c_{\mathrm{ent}}$ 从 0.05 按因子 0.9995 衰减至 0.01。

1.4.9　完整算法

TRPO 的完整算法如算法 3 所示。

Algorithm 3 TRPO 算法

初始化 Actor 参数 θ，Critic 参数 ϕ
for iteration $= 1, 2, \ldots$ **do**
　使用 π_θ 收集轨迹 $\{(s_t, a_t, r_t)\}$，用 Critic 估计 $V_\phi(s_t)$
　用 GAE 计算优势 $\hat{A}_t$，回报目标 $\hat{R}_t = \hat{A}_t + V_\phi(s_t)$
　计算策略梯度 $\boldsymbol{g} = \nabla_\theta L_{\theta_{\mathrm{old}}}(\theta)$
　共轭梯度法求解 $\boldsymbol{F}\boldsymbol{x} = \boldsymbol{g}$，计算步长 $\alpha = \sqrt{2\delta/(\boldsymbol{x}^T\boldsymbol{F}\boldsymbol{x})}$
　回溯线性搜索，更新 $\theta \leftarrow \theta + \alpha \cdot \beta^k \cdot \boldsymbol{x}$
　for $i = 1$ to K **do**
　　更新 Critic：最小化 $\sum_t (V_\phi(s_t) - \hat{R}_t)^2$
　end for
end for

1.4.10　与 DQN 的根本区别

表 1.4 总结了 TRPO 与 DQN 的根本区别。TRPO 作为在策略算法，样本效率天然低于 DQN 等离策略算法，但其信赖域约束提供了更强的理论保证——每次更新都能保证策略性能的单调改进。

表 1.4: DQN 与 TRPO 对比

特性	**DQN**	**TRPO**
方法类别	基于价值 (Value-based)	基于策略 (Policy-based)
学习对象	$Q^*(s,a)$	$\pi_\theta(a\|s)$
策略类型	离策略 (Off-policy)	在策略 (On-policy)
数据复用	经验回放（高复用率）	每次更新后丢弃（低复用率）
动作空间	仅离散	离散/连续均可
更新方式	梯度下降	自然梯度 + 线性搜索
稳定性保证	目标网络 (启发式)	信赖域约束 (理论保证)

1.5　本章小结

本章详细介绍了本项目中使用的三种强化学习算法的原理：**DQN (2013)** 首次将深度神经网络与 Q-Learning 结合，通过经验回放打破数据相关性，实现了从原始像素到控制策略的端到端学习；**DQN (2015, Nature)** 引入目标网络解决训练不稳定问题，配合更深的网络和 Huber 损失进一步提升性能；**TRPO** 从策略优化角度出发，通过信赖域约束和自然梯度保证每次更新的安全性，结合 Actor-Critic 架构、GAE 和熵正则化实现可靠的策略梯度学习。三者分别代表了深度强化学习中基于价值和基于策略的经典范式。

第二章　DQN (2013) 实验

2.1　代码设置

本节概述 DQN 2013 的代码实现与实验配置。算法核心代码位于 `DQN-2013/dqn/` 目录下，包含 Q-Network（`network.py`）、经验回放缓冲区（`replay_buffer.py`）和智能体（`agent.py`）三个模块；游戏环境位于 `envs/` 目录，各算法共用。

2.1.1　游戏环境

实验选用两款 Atari 游戏：**Breakout**（打砖块，4 个离散动作）和 **Video Pinball**（弹球，9 个离散动作）。所有游戏基于 Gymnasium + ALE 框架，继承自统一的 `BaseGameEnv` 基类。环境配置包括跳帧（frameskip = 4）和随机 NOOP 初始化（0~30 次空操作）。

原始帧 $(210 \times 160 \times 3)$ 经过帧间最大化（消除闪烁）、灰度化、下采样至 84×84 的预处理流水线。`ClipRewardEnv` 将奖励裁剪为 $\{-1, 0, +1\}$，`FrameStack` 堆叠最近 4 帧为 $(4, 84, 84)$ 的网络输入。

2.1.2　网络结构与训练

Q-Network 采用原始论文的 CNN 架构：2 层卷积 ($16 \times 8 \times 8$, stride=4 $\rightarrow$ $32 \times 4 \times 4$, stride=2)、1 层 256 维全连接，输出 $|A|$ 个 Q 值。输入 uint8 图像在前向传播中自动归一化至 $[0, 1]$。

经验回放缓冲区采用**单帧存储**策略（仅存 $(84, 84)$ 单帧而非完整 4 帧堆叠），采样时动态拼接，节省约 87.5% 内存。多环境场景下为每个环境维护独立缓冲区，避免跨环境拼帧，并正确处理 episode 边界的零填充。

训练采用 ε-greedy 策略（ε 从 1.0 线性衰减至 0.1）、MSE 损失和 RMSProp 优化器 ($\text{lr} = 0.00025$, $\alpha = 0.95$)。DQN 2013 使用同一个网络计算当前 Q 值和 TD 目标，这是其与 2015 版的核心区别。

2.1.3　默认超参数

表 2.1 列出智能体的默认超参数，后续各实验会在此基础上进行调整。

表 2.1: DQN 2013 默认超参数

参数	默认值	说明
`replay_capacity`	1,000,000	经验回放容量（帧数）
`batch_size`	256	mini-batch 大小
`gamma`	0.99	折扣因子
`learning_rate`	0.00025	RMSProp 学习率
`epsilon_start / end`	1.0 / 0.1	探索率起止值
`epsilon_decay_frames`	1,000,000	ε 衰减帧数
`warmup_frames`	50,000	预热帧数（不训练）

2.1.4　训练基础设施

训练使用 Gymnasium 的 `AsyncVectorEnv` 创建多个异步并行环境（子进程以 `spawn` 方式启动，共享内存传输数据）。每步从所有环境同时收集经验后执行一次梯度更新。硬件为 NVIDIA RTX 4060 Laptop GPU（8 GB 显存），PyTorch 2.9.1 + CUDA 12.8，在本地进行训练。

2.2　实验一：Video Pinball 基线测试

本次实验作为项目的首次尝试，选用 Video Pinball 游戏，采用最基础的配置进行训练，目的是验证整体流程的正确性并建立性能基线。关键参数配置如表 2.2 所示。

表 2.2: 实验一参数配置（与默认值的差异）

参数	本次值	默认值	说明
游戏环境	Video Pinball	—	9 个离散动作
并行环境数	1	—	未使用并行
`replay_capacity`	100,000	1,000,000	缩小为默认的 1/10
`batch_size`	32	256	缩小为默认的 1/8
总训练帧数	10,000,000	—	—

2.2.1　训练结果与分析

图 2.1 展示了训练曲线。左图为每回合奖励及 100 回合滑动平均，右图为 TD Loss。

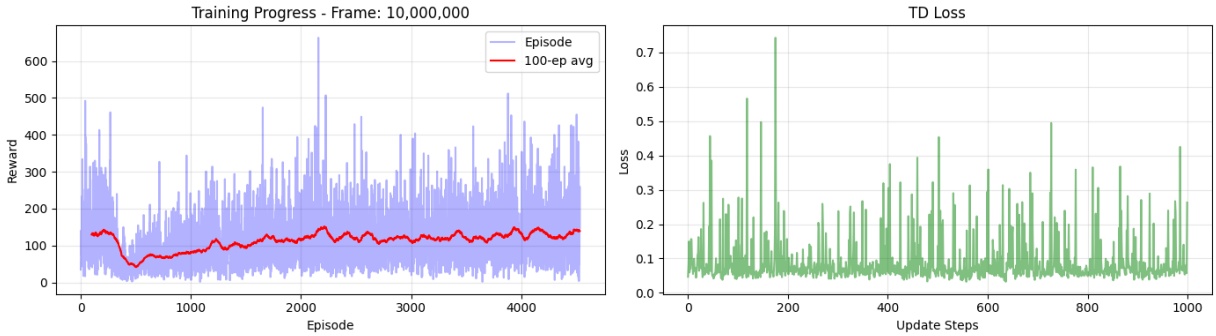

图 2.1: 实验一训练曲线：Video Pinball，单环境，batch_size=32，buffer=100K

从训练曲线可以观察到两个显著问题：

（1）奖励几乎没有提升。100 回合平均奖励在整个训练过程中始终在 100~150 之间徘徊，没有明显的上升趋势。单回合奖励虽然偶有 400~700 的峰值，但这主要来自 Video Pinball 游戏本身的随机性（弹球偶尔击中高分目标），而非智能体学到了有效策略。TD Loss 在 0.05~0.15 之间波动，偶有尖峰但整体无收敛趋势，说明 Q 值估计未能有效改善。

（2）训练速度极慢。由于仅使用单个环境，数据收集完全串行，实测训练速度仅约 300 帧/秒。以此速度完成 10M 帧训练需要近 10 小时，且数据利用效率低下——单环境产生的连续样本高度相关，经验回放缓冲区仅 100K 帧，多样性严重不足。

2.2.2　问题诊断

本次实验表现不佳的原因主要有三点：

（a）回放缓冲区过小。此时的实现采用四帧堆叠的完整格式存储经验，即每条经验的状态为 $(4, 84, 84)$ 的 uint8 数组，单条经验占用 $4 \times 84 \times 84 = 28{,}224$ 字节。若将缓冲区设为论文建议的 1M 帧，仅状态数据就需要约 $28{,}224 \times 10^6 \times 2 \approx 53$ GB 内存（需同时存储当前状态和下一状态），远超本机的可用内存（32G）。因此本次实验被迫将缓冲区限制在 100K 帧，但

这仅能存储约 25 分钟的游戏经验（按 4 帧跳帧计），缓冲区快速被覆盖，早期有价值的经验迅速丢失，采样多样性严重不足。

(b) batch size 过小。32 个样本的 mini-batch 导致梯度估计方差大，Q 值更新不稳定。

(c) 单环境采样。数据收集完全串行，效率低下，且连续帧之间的强相关性未被充分打破。

基于以上分析，后续实验将引入并行环境、改用单帧存储以增大缓冲区容量，并增大 batch size。

2.3　实验二：Video Pinball 并行环境测试

针对实验一中训练速度过慢的问题，本次实验引入并行环境并增大 batch size，同时保持缓冲区容量不变（此时尚未实现单帧存储优化）。参数变化如表 2.3 所示。

表 2.3: 实验二参数配置（与实验一的差异）

参数	本次值	实验一	说明
并行环境数	**8**	1	引入 `AsyncVectorEnv`
`batch_size`	**256**	32	增大 8 倍
`replay_capacity`	100,000	100,000	未变（受四帧存储内存限制）

2.3.1　训练结果与分析

图 2.2 展示了训练曲线。

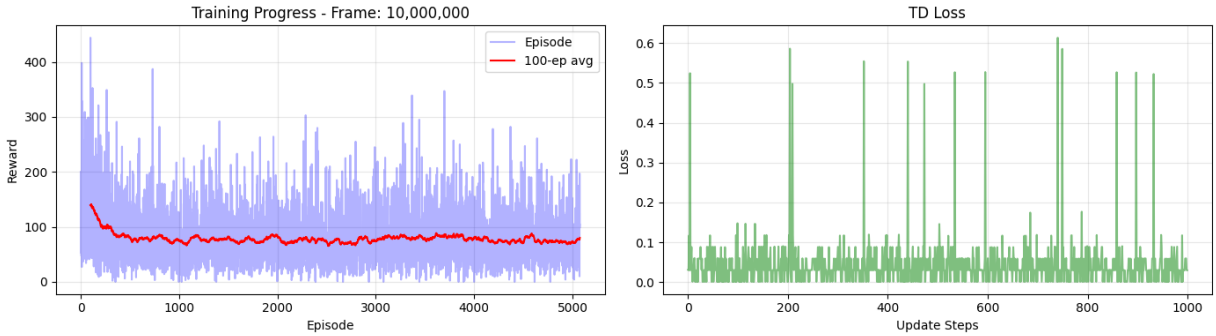

图 2.2: 实验二训练曲线：Video Pinball，8 并行环境，batch_size=256，buffer=100K

与实验一相比，结果呈现出一升一降的特点：

训练速度大幅提升。得益于 8 个并行环境同时采集数据，训练速度从实验一的约 300 帧/秒提升至约 1,200 帧/秒，加速近 4 倍，10M 帧训练耗时从近 10 小时缩短至不到 3 小时。

训练效果反而有所下滑。100 回合平均奖励从实验一的 100~150 下降至 70~90，整体呈轻微下降趋势。这说明仅靠并行环境和增大 batch size 并不能解决根本问题——回放缓冲区仍然只有 100K 帧，而 8 个并行环境使得缓冲区被覆盖的速度加快了 8 倍，经验的有效存留时间进一步缩短。

2.3.2　游戏适配性分析与决策

在分析训练效果不佳的原因时，我们仔细观察了 Video Pinball 的游戏画面，发现该游戏的视觉场景远比预期复杂：弹球台包含大量细小的弹射器、通道和得分区域，弹球运动轨迹快速且不规则。DQN 2013 仅有 2 层卷积（$16 \times 8 \times 8$ 和 $32 \times 4 \times 4$），感受野和特征提取能力有限，难以从 84×84 的灰度图中捕捉足够的视觉信息来学习有效策略。

相比之下，Breakout（打砖块）的视觉元素更加简洁——仅有球、挡板和砖块三类对象，运动模式规律，非常适合作为验证 DQN 算法有效性的基准游戏。因此，**后续实验决定切换至 Breakout 游戏**，以排除游戏复杂度对算法评估的干扰。

2.4　实验三：Breakout 批次大小对比实验

基于实验二的分析，本次实验切换至 Breakout 游戏，并将回放缓冲区从 100K 扩大至 400K（400K 已是当前内存条件下能支撑的最大容量）。为探究批次大小对训练效果的影响，我们设计了两组对照实验，参数如表 2.4 所示，两组均训练 30M 帧。

表 2.4: 实验三对照参数配置

参数	配置 A	配置 B
游戏环境	Breakout	Breakout
并行环境数	4	**8**
`batch_size`	128	**256**
`replay_capacity`	400,000	400,000
总训练帧数	30,000,000	30,000,000

2.4.1　训练结果对比

图 2.3 和图 2.4 分别展示了两组配置的训练曲线。

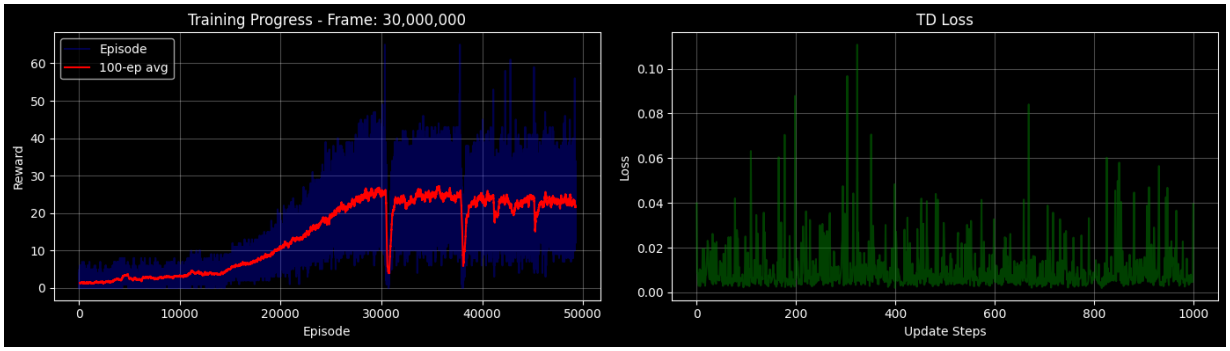

图 2.3: 配置 A：Breakout，4 并行环境，batch_size=128，buffer=400K

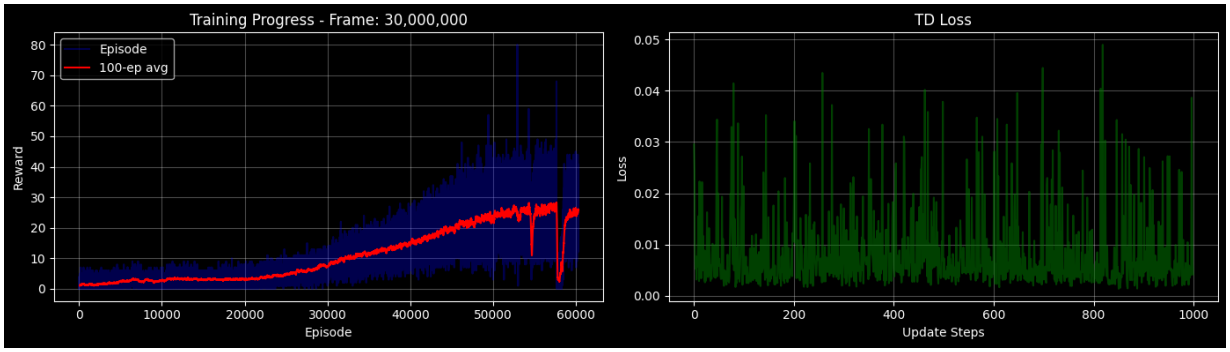

图 2.4: 配置 B：Breakout，8 并行环境，batch_size=256，buffer=400K

两组实验均首次在 Breakout 上展现出明显的学习效果，100 回合平均奖励最终都达到约 25 分，相较于 Video Pinball 实验有质的突破。但两组在训练过程中呈现出显著差异：

配置 A（batch=128）：学习速度快但不稳定。奖励在约 25K 回合处出现一次急剧跳升（从 ~8 跃升至 ~27），但随后在 30K 回合附近发生明显的性能塌陷（骤降至 ~5），经过一段时间才恢复至 20~25。TD Loss 波动较大，后期出现 0.08~0.10 的尖峰，反映出 Q 值估计的不稳定。

配置 B（batch=256）：学习更平稳且后期更优。奖励曲线呈现渐进式上升，无急剧跳变，虽然在 55K 回合处也出现一次短暂回落，但幅度远小于配置 A，且迅速恢复。TD Loss 整体维持在 0.01~0.03 的较低水平，仅为配置 A 的约 1/3，训练过程显著更平稳。

2.4.2　分析

以上对比结果符合大批次训练的经典特性：较大的 mini-batch 提供了更准确的梯度估计，降低了更新方差，使 Q 值收敛过程更加平稳；代价是每次梯度更新需要更多样本，表观学习速度稍慢。但从最终性能和训练稳定性来看，batch_size=256 明显优于 128。因此，**后续实验统一采用 8 并行环境、batch_size=256 的配置**。

2.5　实验四：缓冲区容量对训练效果的影响

实验三使用了 400K 的回放缓冲区（已耗尽可用内存），取得了较好的训练效果。为进一步验证缓冲区容量对训练的重要性，本次实验在保持 8 并行环境、batch_size=256 的最优配置下，将缓冲区缩小回 100K，分别训练 10M 帧和 30M 帧，观察较小缓冲区下的训练表现。参数如表 2.5 所示。

表 2.5： 实验四参数配置

参数	本次值	实验三配置 B	说明
并行环境数	8	8	不变
`batch_size`	256	256	不变
`replay_capacity`	**100,000**	400,000	缩小为 1/4
总训练帧数	10M / 30M	30M	两组对比

2.5.1　训练结果

图 2.5 和图 2.6 分别展示了 10M 帧和 30M 帧的训练曲线。

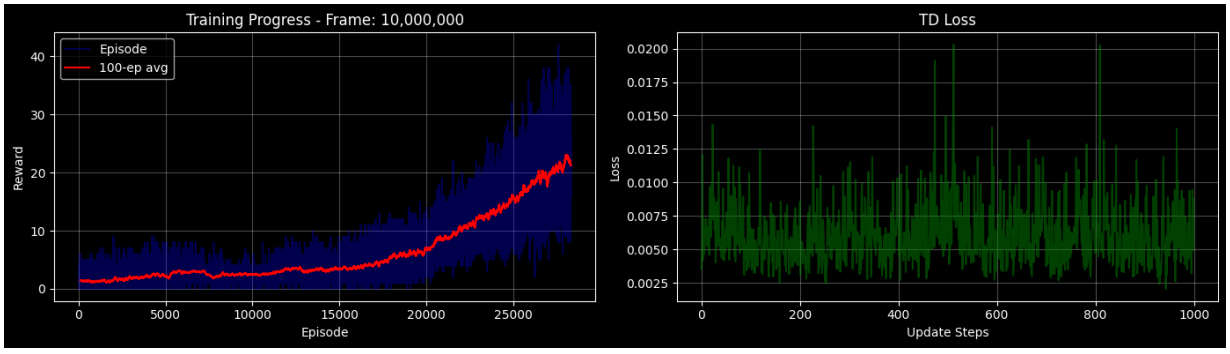

图 2.5： 实验四（10M 帧）：Breakout，8 并行环境，batch_size=256，buffer=100K

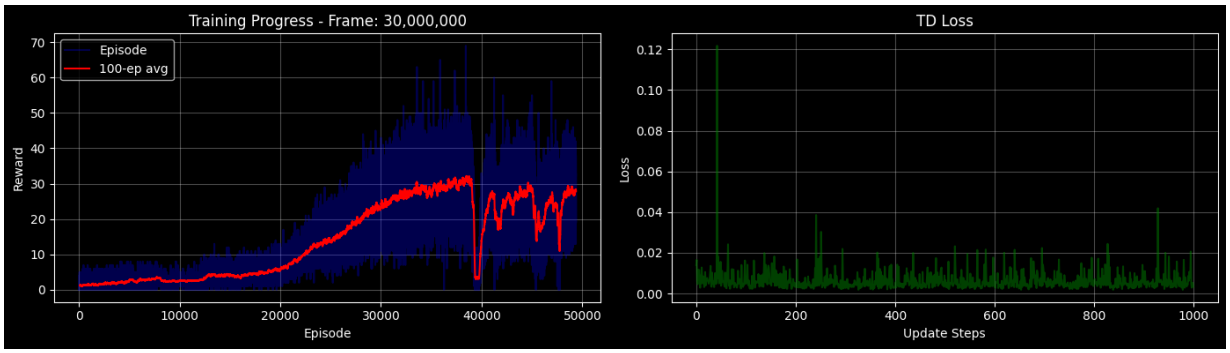

图 2.6： 实验四（30M 帧）：Breakout，8 并行环境，batch_size=256，buffer=100K

2.5.2　分析

10M 帧时，100 回合平均奖励从 ∼2 缓慢上升至 ∼21，曲线末段仍有上升趋势，TD Loss 维持在 0.003~0.015 的低水平，训练过程看似正常。

30M 帧时，奖励进一步上升至 ∼30，但在约 40K 回合处发生严重的性能塌陷（骤降至 ∼5），之后虽恢复至 ∼28，但训练过程的不稳定性已充分暴露。与实验三配置 B（400K 缓冲区，同样 30M 帧）相比，虽然峰值奖励略高，但塌陷幅度更大、恢复更慢，整体稳定性明显不足。

两组结果共同反映出**缓冲区过小对长时间训练的负面影响**：8 个并行环境每秒产生大量新经验，100K 的缓冲区仅能容纳约 100,000/8 = 12,500 步的经验（约 50 秒的游戏时间），旧经验被快速覆盖，导致采样多样性不足。训练初期由于 ε 较大、策略变化快，新经验本身具有一定多样性，问题尚不突出；但随着训练深入，策略趋于稳定，缓冲区中充斥着高度相似的近期经验，Q 值估计出现过拟合，最终引发性能塌陷。而实验三的 400K 缓冲区能存储 4 倍的经验，有效缓解了这一问题。这一对比充分说明了**充足的回放缓冲区容量对 DQN 训练稳定性的关键作用**。

2.6　本章小结

通过四组实验，我们系统地探索了 DQN 2013 在 Atari 游戏上的训练表现。从最初 Video Pinball 上几乎无效的基线（实验一、二），到切换 Breakout 后首次观察到明显的学习效果（实验三），再到缓冲区容量对比实验（实验四），我们逐步确定了较优的训练配置：8 并行环境、batch_size=256、400K 回放缓冲区。

然而，即便在最优配置下，DQN 2013 的 100 回合平均奖励也仅稳定在 25 分左右，且训练过程中仍会出现性能塌陷现象。这一瓶颈的根本原因在于 DQN 2013 的固有缺陷——**同一网络同时用于动作选择和 TD 目标计算**，参数每次更新后 TD 目标随之改变，造成“追逐移动目标”的困境，长时间训练后 Q 值估计容易发散。此外，2 层卷积网络的表达能力也限制了策略的上限。

鉴于 DQN 2013 的性能已难以进一步提升，我们决定转向 **DQN 2015 (Nature DQN)**，通过引入目标网络、更深的网络结构和 Huber 损失等改进来突破当前瓶颈。

第三章　DQN (2015) 实验

3.1　代码设置

本章实验基于 DQN 2015（Nature DQN）实现，代码位于 `DQN-2015/dqn/` 目录，整体结构与 DQN 2013 一致（Q-Network、经验回放缓冲区、智能体三个模块），但在网络结构、训练机制和超参数上进行了关键改进。实验统一使用 Breakout 游戏。

3.1.1　网络结构改进

Q-Network 从 2 层卷积扩展为 3 层卷积，全连接层宽度从 256 增至 512，参数量从约 1.7M 增至约 3.5M。特征图尺寸变化为 $4\times84\times84 \xrightarrow{32\times8\times8,s=4} 32\times20\times20 \xrightarrow{64\times4\times4,s=2} 64\times9\times9 \xrightarrow{64\times3\times3,s=1} 64\times7\times7 \xrightarrow{\text{Flatten}} 3136 \xrightarrow{\text{FC}} 512 \to |A|$。更深的网络赋予了模型更强的视觉特征提取能力。

3.1.2　目标网络机制

DQN 2015 的核心改进是引入**目标网络**（Target Network）。主网络参数 θ 用于动作选择和梯度更新，目标网络参数 θ^- 仅用于计算 TD 目标 $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$，不参与梯度计算。目标网络每隔 $C = 10{,}000$ 步从主网络硬拷贝一次（$\theta^- \leftarrow \theta$），在两次同步之间 TD 目标保持稳定，从根本上解决了 DQN 2013 中“追逐移动目标”的问题。

3.1.3　损失函数与梯度裁剪

损失函数从 MSE 改为 **Huber 损失**（PyTorch 中的 `smooth_l1_loss`），小误差区域保持二次函数的快速收敛，大误差区域退化为线性函数，有效抑制异常 TD 误差导致的梯度爆炸。同时引入梯度裁剪（$\|\nabla\|_{\max} = 10.0$），进一步保障训练稳定性。

3.1.4　默认超参数

表 3.1 列出 DQN 2015 的默认超参数。与 DQN 2013 相比，主要变化为新增目标网络同步频率和梯度裁剪，ε 衰减帧数延长至 10M 以适应更长的训练周期。经验回放缓冲区沿用单帧存储策略，容量提升至 1M 帧。

3.1.5　训练基础设施

训练基础设施沿用第二章的并行环境框架（8 个 `AsyncVectorEnv`），总训练帧数延长至 50M。经验回放缓冲区采用单帧存储策略，1M 帧容量下内存占用约为 $84 \times 84 \times 10^6 \approx 6.7$ GB，在可用内存范围内。

3.2　实验一：小缓冲区基线测试

本次实验作为 DQN 2015 的首次尝试，沿用了第二章中 DQN 2013 的经验回放实现（四帧堆叠完整存储），因此缓冲区容量仍被限制在 100K 帧。参数如表 3.2 所示。

表 3.1: DQN 2015 默认超参数

参数	默认值	说明
`replay_capacity`	1,000,000	经验回放容量（单帧存储）
`batch_size`	256	mini-batch 大小
`gamma`	0.99	折扣因子
`learning_rate`	0.00025	RMSProp 学习率
`epsilon_start / end`	1.0 / 0.1	探索率起止值
`epsilon_decay_frames`	10,000,000	ε 衰减帧数
`warmup_frames`	50,000	预热帧数
`target_update_freq`	10,000	目标网络同步频率
`grad_clip`	10.0	梯度裁剪阈值

表 3.2: DQN 2015 实验一参数配置

参数	本次值	说明
并行环境数	8	`AsyncVectorEnv`
`batch_size`	256	—
`replay_capacity`	100,000	受四帧存储内存限制
总训练帧数	10,000,000	—

3.2.1　训练结果与分析

图 3.1 展示了训练曲线。

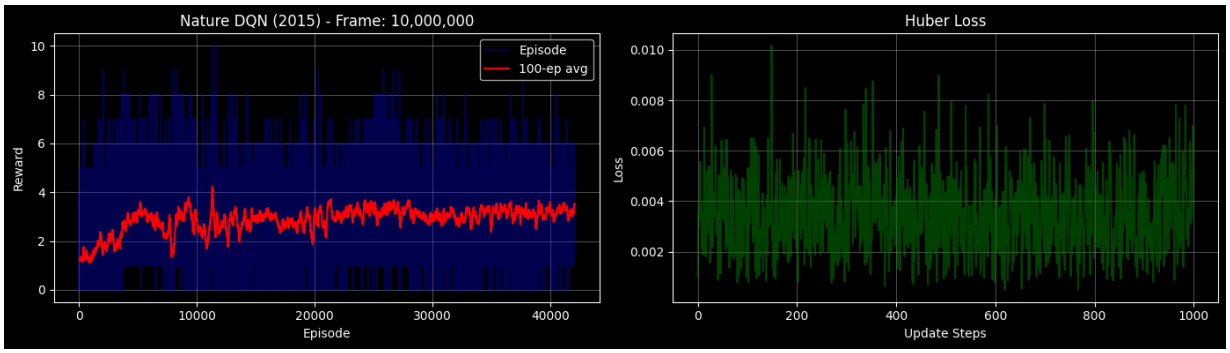

图 3.1: DQN 2015 实验一：Breakout，8 并行环境，batch_size=256，buffer=100K

训练效果极差：100 回合平均奖励在整个 10M 帧训练过程中始终徘徊在 2~3.5，几乎没有学习迹象。单回合奖励偶尔达到 7~9，但绝大多数在 0~4 之间。Huber Loss 在 0.002~0.008 之间波动，无收敛趋势。这一表现甚至不如第二章中 DQN 2013 在相同缓冲区容量下的结果（实验四，avg ~21）。

3.2.2　问题诊断

DQN 2015 在小缓冲区下表现反而不如 DQN 2013，原因在于目标网络机制对缓冲区容量有更高的要求。DQN 2013 中 TD 目标随主网络实时变化，虽然不稳定，但至少能跟踪当前策略的最新经验；而 DQN 2015 的目标网络每 10,000 步才同步一次，在此期间 TD 目标是基于旧参数计算的。若缓冲区过小（100K 帧，8 并行环境下仅约 50 秒经验），目标网络同步时缓冲区中的经验可能已与当前策略严重脱节，导致 TD 目标既“旧”（来自过时的目标网络）又“窄”（缓冲区中缺乏多样性），Q 值更新方向产生系统性偏差，训练无法收敛。

这一结果表明，**充足的缓冲区容量是 DQN 2015 发挥目标网络优势的前提**。为此，我们对经验回放缓冲区进行了重构，改为**单帧存储**策略：每条经验仅存储单帧 $(84, 84)$ 而非完整的 4 帧堆叠 $(4, 84, 84)$，采样时动态拼接相邻 4 帧，并正确处理 episode 边界的零填充。这

一优化将单条经验的状态存储量降低至原来的 1/4，使得在相同内存条件下缓冲区容量可扩展至 **1,000,000 帧**，与原始论文的配置一致。

3.3　实验二：1M 缓冲区首次训练（Version 1）

实现单帧存储优化后，缓冲区容量成功扩展至 1M 帧。本次实验使用完整的 Nature DQN 配置进行约 46M 帧的长时间训练。参数如表 3.3 所示。

表 3.3: DQN 2015 实验二参数配置

参数	本次值	说明
`replay_capacity`	**1,000,000**	单帧存储，与论文一致
`epsilon_decay_frames`	1,000,000	仅占总帧数的 2%
`grad_clip`	无	未启用梯度裁剪
RMSProp momentum	0.95	非论文标准配置
总训练帧数	~46,000,000	—

3.3.1　训练结果

图 3.2 展示了训练曲线。

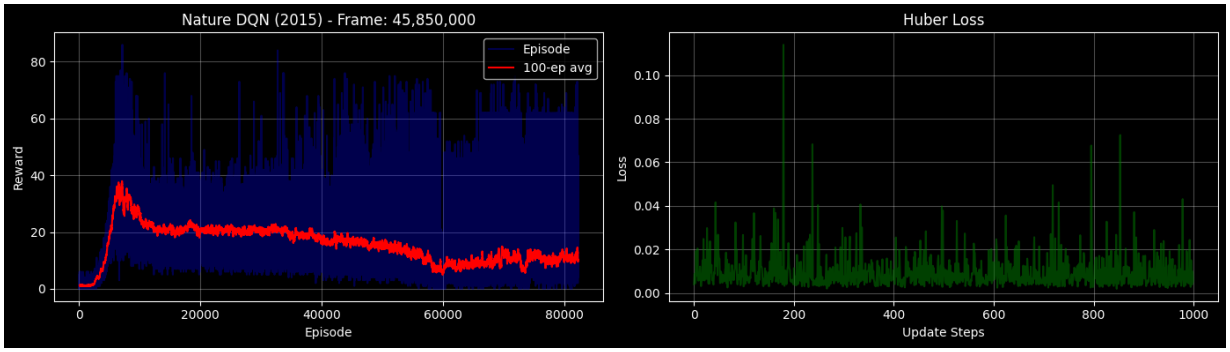

图 3.2: DQN 2015 Version 1：Breakout，1M 缓冲区，ε 衰减 1M 帧，无梯度裁剪

训练曲线呈现出典型的“先升后降”模式：前 15K 回合奖励快速上升至 ~25~35，展现出较强的学习能力；但在 15K~40K 回合进入平台期后，奖励开始持续下滑，最终在 80K 回合时衰退至 ~10，性能从峰值下降约 60%。Huber Loss 整体在 0.01~0.02 之间，但后期出现高达 0.10 的剧烈尖峰，表明梯度爆炸正在破坏已学得的策略。

3.3.2　问题诊断

分析发现三个关键缺陷：

(a) 缺少梯度裁剪。训练过程中偶发的大 TD 误差产生极大梯度，一次性大幅修改网络参数，破坏已学好的策略。Loss 曲线后期的尖峰正是梯度爆炸的直接体现，这是导致训练后期崩溃的最主要原因。

(b) 探索率衰减过快。ε 在仅 1M 帧（总训练量的 2%）内即从 1.0 衰减至 0.1，意味着后 98% 的训练几乎完全依赖利用。回放缓冲区逐渐被大量相似的次优经验填满，多样性严重不足，引发灾难性遗忘。

(c) RMSProp 配置偏离论文。额外的 `momentum=0.95` 引入了过大的优化惯性，可能导致参数更新过冲，加剧训练不稳定。

3.4　实验三：修复后的正式训练（Version 2）

针对实验二暴露的三个问题，我们对代码进行了修复，修改内容如表 3.4 所示。

表 3.4: Version 1 → Version 2 修复内容

配置项	Version 1	Version 2
梯度裁剪	无	$\|\nabla\|_{\max} = 10.0$
`epsilon_decay_frames`	1,000,000	**10,000,000**
RMSProp momentum	0.95	**移除**

3.4.1　训练结果

图 3.3 展示了修复后的训练曲线，训练完整进行了 50M 帧。

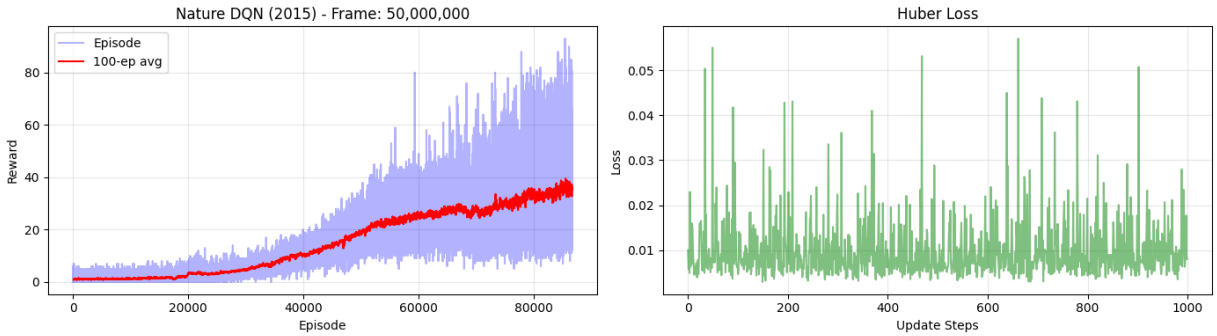

图 3.3: DQN 2015 Version 2：Breakout，1M 缓冲区，ε 衰减 10M 帧，梯度裁剪 10.0

与 Version 1 形成鲜明对比，Version 2 的训练曲线呈现出**稳定的单调上升**趋势：100 回合平均奖励从 ~2 持续增长至 ~35，全程无性能塌陷现象。Huber Loss 稳定在 0.005~0.015 之间，偶有 0.03~0.05 的小尖峰但远不及 Version 1 的 0.10 量级，梯度裁剪有效抑制了梯度爆炸。

3.4.2　改进效果分析

三项修复的效果可从训练曲线中清晰辨识：

梯度裁剪消除了 Loss 的剧烈尖峰，防止了偶发大梯度对策略的破坏，是训练后期不再崩溃的最关键因素。

延长 ε 衰减使探索贯穿前 10M 帧（总量的 20%），曲线前期上升虽较 Version 1 缓慢（因高 ε 下随机动作比例大），但回放缓冲区始终保持足够的经验多样性，为后期稳定学习奠定了基础。

移除 RMSProp momentum 使优化行为回归论文标准配置，参数更新更加可控，配合梯度裁剪进一步提升了训练稳定性。

最终 Version 2 的 100 回合平均奖励达到 ~35，不仅远超 Version 1 的最终表现（~10），也显著超越了第二章中 DQN 2013 的最佳结果（~25），充分体现了目标网络、更深网络和 Huber 损失等改进在充足缓冲区支撑下的优势。

3.5　本章小结

通过三组实验，我们验证了 DQN 2015 相较于 DQN 2013 的显著优势。实验一暴露了目标网络对缓冲区容量的高要求，促使我们实现了单帧存储优化，将缓冲区扩展至 1M 帧。实验二在大缓冲区下展现出强学习能力但因梯度裁剪缺失等配置问题导致训练后期崩溃。实验三修复三项关键缺陷后，训练全程稳定上升，最终达到 ~35 的平均奖励，验证了 Nature DQN 的有效性。

第四章　TRPO 实验

4.1　代码设置

本章实验基于 TRPO（Trust Region Policy Optimization）实现，代码位于 `TRPO/trpo/` 目录。与前两章的 DQN（值函数方法）不同，TRPO 是一种**策略梯度方法**，直接优化参数化策略 π_θ，并通过信赖域约束保证每次更新的稳定性。实验均使用 Breakout 游戏。

4.1.1　网络结构

TRPO 采用 Actor-Critic 架构，策略网络（Actor）和价值网络（Critic）各自拥有独立的 CNN 骨干（结构与 DQN 2015 相同：3 层卷积 + 全连接），不共享参数。Actor 输出各动作的 log 概率（log-softmax），Critic 输出标量状态价值 $V(s)$。两个网络均使用正交初始化，其中 Actor 输出层增益为 0.01（保证初始策略接近均匀分布），Critic 输出层增益为 1.0。

4.1.2　TRPO 更新机制

每次策略更新包含以下步骤：**(1)** 收集一批 on-policy 轨迹数据，计算 GAE 优势估计；**(2)** 计算策略梯度 $g = \nabla_\theta L(\theta)$，其中 L 包含策略目标和熵正则项；**(3)** 通过共轭梯度法求解自然梯度 $F^{-1}g$（Fisher 信息矩阵通过 Hessian-向量积隐式计算，避免显式构造）；**(4)** 沿自然梯度方向进行线性搜索，找到满足 KL 散度约束 $D_{\mathrm{KL}}(\pi_{\mathrm{old}} \| \pi_{\mathrm{new}}) \le \delta$ 且性能有提升的最大步长；**(5)** 使用 Adam 优化器独立训练 Critic 网络。

4.1.3　关键区别：On-Policy vs Off-Policy

TRPO 与 DQN 的根本区别在于数据使用方式。DQN 是 off-policy 方法，使用经验回放缓冲区存储历史经验，每条经验可被多次采样训练；TRPO 是 on-policy 方法，**每次策略更新后必须丢弃所有旧数据**，仅使用当前策略采集的新数据。这意味着 TRPO 的样本利用效率远低于 DQN——10M 步训练中，DQN 可执行数百万次网络更新（从 1M 缓冲区反复采样），而 TRPO 仅能进行数百至数千次策略更新。

4.1.4　版本演进

我们先后实现了两个版本：**Version 1**（基础版，`train.ipynb`）采用单环境采样，保守的信赖域配置；**Version 2**（优化版，`train_optimized.py`）引入 8 并行环境、奖励归一化、探索增强和自适应稳定机制。两版的关键超参数对比如表 4.1 所示。

4.2　实验一：Version 1 基础训练

图 4.1 展示了 Version 1 的训练曲线（4 子图分别为回合奖励、评估奖励、KL 散度和训练损失）。

表 4.1: TRPO Version 1 vs Version 2 关键超参数

参数	**Version 1**	**Version 2**	**说明**
并行环境数	1	**8**	SubprocVecEnv
`rollout_steps`	2,048	4,096	每次采集步数
每次更新样本量	2,048	**32,768**	×16
`max_kl`	0.01	**0.02**	信赖域放宽
`damping`	0.1	**0.05**	阻尼减半
`entropy_coef`	0.01	**0.05**→0.01	5 倍探索 + 衰减
`critic_train_iters`	5	**10**	Critic 训练加倍
`gae_lambda`	0.95	**0.98**	更长期优势估计
奖励归一化	无	**有**	RunningMeanStd
自适应稳定	无	**有**	动态调整 max_kl
总训练帧数	10,000,000	10,000,000	—

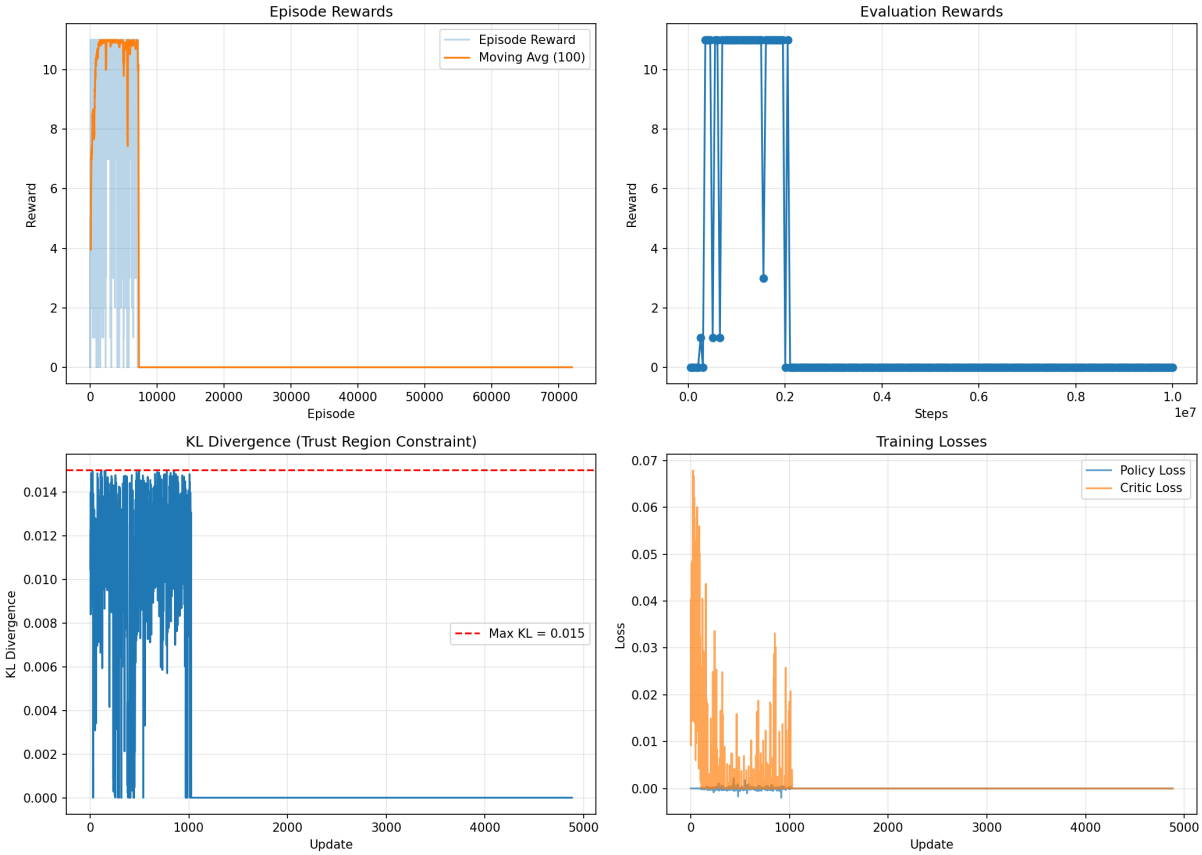

图 4.1: TRPO Version 1：单环境，max_kl=0.01，entropy_coef=0.01

训练呈现典型的“先学后崩”模式。前 10K 回合中，奖励迅速上升至 ~11（学会了基本的接球和击砖动作），评估奖励也一度达到 ~11。但在约 10K 回合处，奖励骤降至 0 并再未恢复，后续 60K 回合的奖励始终为 0。

KL 散度曲线和损失曲线揭示了崩溃的内在机制：在前 ~1,000 次更新中，KL 散度在 0~0.014 之间正常波动，说明策略更新在进行；但此后 KL 散度突降至 0，策略损失和 Critic 损失也同步归零。这意味着**策略已退化为确定性分布**（某个动作概率趋近 1.0），新旧策略完全相同，无法产生有效更新，训练进入不可逆的“死亡状态”。

4.3　实验二：Version 2 优化训练

针对 Version 1 的问题，Version 2 进行了全面优化：8 并行环境将每次更新样本量提升 16 倍，熵系数提高至 0.05 以增强探索，信赖域放宽至 max_kl=0.02，并引入奖励归一化和自适应稳定机制。图 4.2 展示了训练曲线。

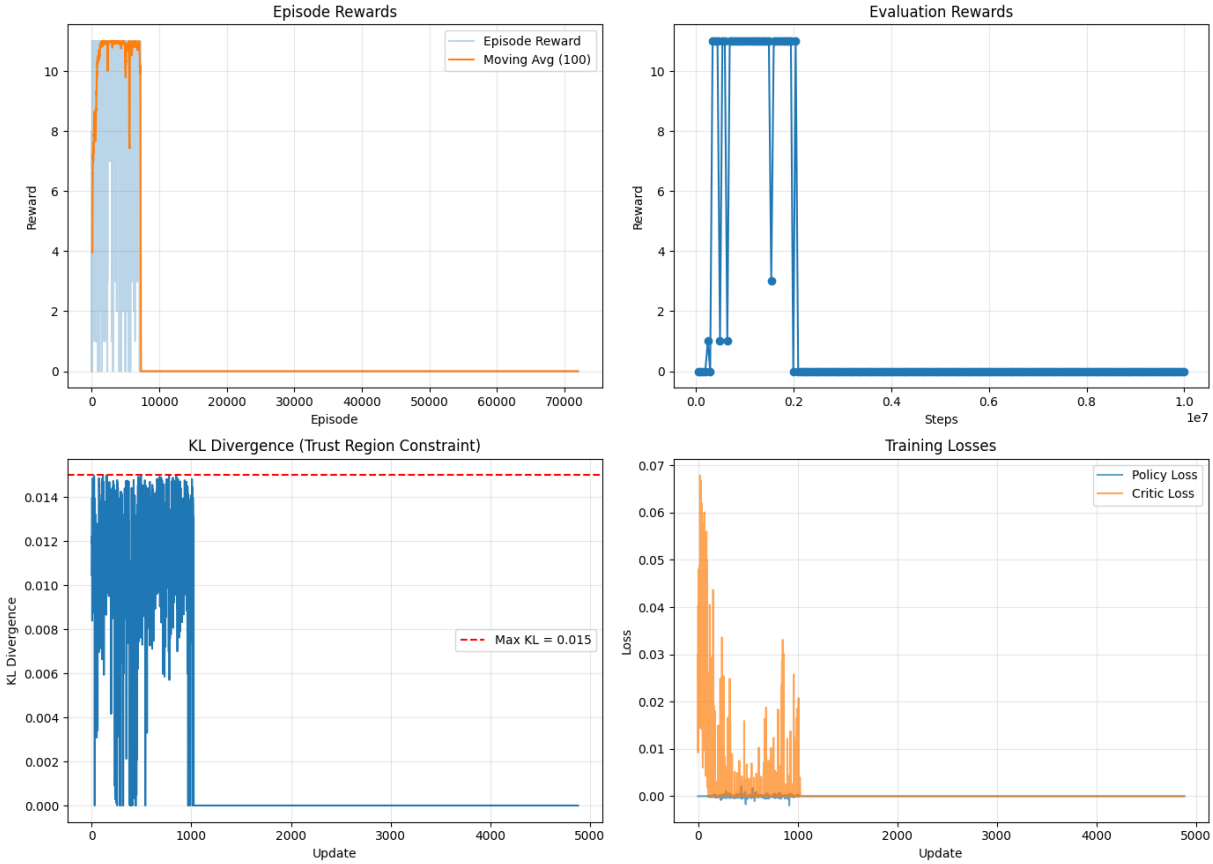

图 4.2: TRPO Version 2：8 并行环境，max_kl=0.02，entropy_coef=0.05→0.01

遗憾的是，Version 2 的训练曲线呈现出与 Version 1 极为相似的模式：奖励在前 ~8K 回合上升至 ~11，随后在 ~10K 回合处再次崩溃至 0，之后的 60K 回合同样无法恢复。KL 散度在前 ~1,000 次更新中一度频繁触及 max_kl=0.015 的上限，说明策略更新较为激进，但随后同样降至 0。Critic 损失从初始的 ~0.07 逐步下降，在崩溃后归零。

Version 2 相比 Version 1 的改进体现在：(1) 训练前期的奖励上升更快、更稳定，得益于更大的样本量和更强的探索；(2) Critic 损失的初期下降更加平滑。但核心的**熵坍缩**问题并未解决——尽管初始熵系数提高了 5 倍，衰减机制（$0.9995^{1000} \approx 0.61$）使得在崩溃发生前熵系数已从 0.05 衰减至 ~0.03，未能阻止策略退化。

4.4 失败原因深度分析

两个版本的 TRPO 在 Breakout 上均以失败告终，最终奖励为 0，远不如 DQN 2013（∼25）和 DQN 2015（∼35）。我们从多个层面分析失败原因。

4.4.1 直接原因：熵坍缩与策略死亡

两个版本的训练曲线都在 KL 散度降至 0 时同步崩溃，这是**熵坍缩**（entropy collapse）的典型表现。当策略对某个动作的概率趋近 1.0 时，策略熵趋近 0，log 概率趋近极端值。此时：(a) 新旧策略几乎相同，KL 散度为 0，信赖域约束无意义；(b) 策略梯度趋近 0（因为 $\nabla \log \pi$ 在确定性分布处梯度消失）；(c) TRPO 的线性搜索无法找到改善方向，所有更新被拒绝；(d) on-policy 采样在退化策略下只能收集到单一动作的经验，无法自我纠正。一旦进入这一状态，训练将永久停滞。

4.4.2 根本原因：On-Policy 方法在高维视觉任务上的固有困难

(1) 样本效率极低。TRPO 每次更新后丢弃全部数据，Version 2 在 10M 步中仅进行约 305 次策略更新（每次 32K 样本），而 DQN 2015 可进行数百万次 Q 值更新。在 Atari 这种需要大量经验才能学到有效视觉特征的任务中，TRPO 的数据利用率严重不足。

(2) 错误不可逆。DQN 的经验回放缓冲区保留了大量历史经验，即使当前策略出现偏差，旧经验仍可纠正 Q 值估计。TRPO 完全依赖当前策略采集数据，一旦策略退化，采集到的数据本身就是有偏的，形成恶性循环。

(3) 视觉特征学习困难。Actor 和 Critic 各自拥有独立的 CNN，需要分别从 84×84 的灰度图中学习有效特征。在仅有数百次策略更新的条件下，CNN 很难学到足够好的视觉表示来支撑策略改进。DQN 通过数百万次梯度更新逐步打磨 CNN 特征，TRPO 不具备这一条件。

(4) 信赖域约束与探索的矛盾。TRPO 的 KL 约束限制了策略每次更新的幅度，这在连续控制任务（如 MuJoCo）中可有效防止策略崩溃。但在 Atari 中，智能体需要大幅度的策略变化才能从“随机挥动”过渡到“精确控制”，保守的信赖域反而限制了学习速度。当学习速度不足以在熵坍缩前建立有效策略时，训练就会崩溃。

4.4.3 实现层面的问题

除算法固有困难外，代码实现中也存在加剧失败的因素：**(a) 线性搜索目标不一致**——Version 1 中策略梯度使用了 $L_{\text{policy}} + \alpha H(\pi)$ 作为优化目标，但线性搜索验收时仅检验 L_{policy}，导致更新方向和验收标准矛盾，线性搜索频繁失败（Version 2 已修复）；**(b) 失败信号掩盖**——线性搜索失败时 KL 散度记录为 0 而非 NaN，在监控中伪装为“稳定更新”，延误了问题发现（Version 2 已修复）。

4.4.4 与 DQN 的对比总结

表 4.2: TRPO 与 DQN 在 Breakout 上的对比

维度	TRPO	DQN 2013	DQN 2015
方法类别	策略梯度（on-policy）	值函数（off-policy）	值函数（off-policy）
数据复用	用后即弃	缓冲区多次采样	缓冲区多次采样
10M 步更新次数	∼305	∼ 数百万	∼ 数百万
错误恢复能力	差（恶性循环）	较好（历史经验纠偏）	好
最终 avg 奖励	∼0	∼25	∼35
训练稳定性	极差（熵坍缩）	一般（偶有塌陷）	好

TRPO 在 Breakout 上的失败并非个例，而是反映了纯 on-policy 策略梯度方法在高维视觉 RL 任务上的系统性困难。在实际应用中，PPO（Proximal Policy Optimization）通过多次利用同一批数据和裁剪目标函数，在一定程度上缓解了样本效率问题，是更常用的策略梯度方法。

第五章　总结与比较

5.1　三种算法实验结果总览

表 5.1 汇总了三种算法在 Breakout 上的最佳实验结果。

表 5.1: 三种算法最佳实验结果对比（Breakout）

	DQN 2013	**DQN 2015**	**TRPO**
最佳 100-ep avg 奖励	∼25	∼35	∼11
最终稳定奖励	∼25	∼35	∼0（崩溃）
训练帧数	30M	50M	10M
训练是否稳定	偶有塌陷	稳定上升	崩溃不可恢复
最优并行环境数	8	8	8
回放缓冲区 / 样本来源	400K（off-policy）	1M（off-policy）	32K/更新（on-policy）
核心网络	2 层 CNN	3 层 CNN	3 层 CNN ×2

5.2　算法特性对比分析

5.2.1　样本效率

样本效率是三种算法表现差异的最主要因素。DQN 系列作为 off-policy 方法，通过经验回放缓冲区实现数据的多次复用：DQN 2013 在 400K 缓冲区中反复采样，DQN 2015 在 1M 缓冲区中反复采样，每一帧经验平均被利用数十次。TRPO 作为 on-policy 方法，每次策略更新后必须丢弃全部数据，10M 帧训练中仅完成约 305 次策略更新。在 Atari 这种需要大量视觉经验才能学到有效特征的任务中，样本效率的差距直接决定了最终性能。

5.2.2　训练稳定性

DQN 2015 的训练最为稳定，得益于三重保护机制：目标网络使 TD 目标在两次同步之间保持恒定，避免了“追逐移动目标”问题；Huber 损失抑制了异常 TD 误差的影响；梯度裁剪防止了偶发的梯度爆炸。DQN 2013 缺少目标网络，训练过程中偶有性能塌陷（如实验三中 batch=128 配置的崩溃），但总体能恢复。TRPO 的训练最不稳定——虽然信赖域约束在理论上保证了单调改进，但在高维视觉任务中，熵坍缩导致策略退化为确定性分布后训练永久停滞，信赖域约束反而成为恢复的障碍。

5.2.3　网络结构的影响

DQN 2013 的 2 层卷积网络（∼1.7M 参数）在视觉特征提取上存在明显局限，这在 Video Pinball 实验中尤为突出——复杂的弹球台场景超出了浅层 CNN 的表达能力。DQN 2015 增加至 3 层卷积和 512 全连接（∼3.5M 参数），配合更多的训练更新次数，能够学到更丰富的视觉表示。TRPO 虽然也使用 3 层 CNN，但 on-policy 更新次数过少，CNN 未能充分训练。

5.3　实验过程中的关键经验

通过本项目的全部实验，我们总结出以下关键经验：

（1）经验回放缓冲区容量至关重要。在 DQN 2013 和 DQN 2015 的实验中，缓冲区过小（100K）始终是训练效果不佳的主要原因。实现单帧存储优化（内存降至 1/4）后将缓冲区扩展至 400K 乃至 1M，是性能提升的转折点。

（2）超参数配置需要与算法特性匹配。DQN 2015 的目标网络对缓冲区容量要求更高（实验表明 100K 缓冲区下甚至不如 DQN 2013）；ε 衰减帧数需与总训练量匹配（1M 帧衰减 vs 10M 帧衰减的巨大差异）；梯度裁剪对长时间训练的稳定性不可或缺。

（3）并行环境是实用性的基础。从单环境的 ∼400 帧/秒到 8 并行环境的 ∼1,200 帧/秒，训练效率提升约 3~4 倍，同时并行采样也提高了数据多样性，改善了训练稳定性。

（4）算法选择需考虑任务特性。TRPO 在 Atari 视觉任务上的失败表明，并非所有理论优美的算法都适合所有任务。On-policy 方法的低样本效率在需要大量视觉经验的任务中是致命的；off-policy 方法通过经验复用在此类任务中具有天然优势。

5.4　未来改进方向

基于本项目的实验结果，可能的改进方向包括：

DQN 方面：引入 Double DQN 缓解 Q 值过估计、Dueling DQN 分离状态价值和动作优势、优先经验回放（Prioritized Experience Replay）提高重要经验的采样频率，以及 Rainbow 等组合方法。

策略梯度方面：将 TRPO 替换为 PPO（Proximal Policy Optimization），PPO 通过裁剪目标函数实现类似的信赖域效果，但允许在同一批数据上进行多次梯度更新，显著提高样本效率。此外，可尝试 Actor-Critic 共享 CNN 骨干以减少参数量并加速特征学习。

工程优化方面：进一步优化内存管理以支持更大的缓冲区、引入混合精度训练加速 GPU 计算、实现分布式训练以扩展并行环境规模。