PG
以ALE LUNARLANDER环境为例

强化学习实验报告

张恒硕

南開大學
Nankai University

目 录

图 片

返回目录

表　格

返回目录

1 实验目的

1. 分析学习LunarLander游戏环境的离散版本和连续版本，了解其状态、动作、奖励等设定。
2. 了解策略梯度（Gradient Pendulm）算法及演员-评论家（Actor-Critic）算法的原理。
3. 编写REINFORCE算法、PPO算法和DDPG算法的代码训练智能体分别进行LunarLander游戏的离散和连续环境，观察训练过程和结果，并对比分析。

2 实验原理

2.1 LunarLander游戏环境

LunarLander是Gym中的经典环境，模拟阿波罗登月舱着陆过程。在该环境中，智能体需要控制登月舱安全平稳地降落在指定区域。以下图1展示了游戏过程。

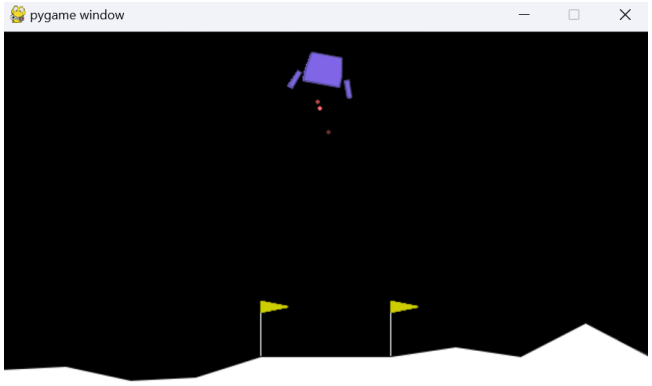


图 1: LunarLander游戏截图

环境分为离散和连续两个版本，分别对应LunarLander和LunarLanderContinuous，但是它们除了动作空间以外，基本一致。

状态空间 8个连续变量：

- (x, y)：登月舱位置坐标。

返回目录

- (v_x, v_y)：登月舱速度分量。
- θ：登月舱倾角。
- ω：登月舱角速度。
- left_leg, right_leg：着陆腿是否接触地面（布尔值，0或1）。

离散动作空间 4种离散动作：

1. 不做任何操作。
2. 开启主发动机（向下推力）。
3. 开启左侧发动机（向右推力）。
4. 开启右侧发动机（向左推力）。

连续动作空间 2维连续动作：

1. 主引擎推力控制，范围$[-1, 1]$
   - 负值无效，等同于0（不使用主引擎）。
   - 正值表示主引擎推力大小（$0-100\%$）。
2. 侧向引擎控制，范围$[-1, 1]$
   - 负值：向左推力（$0-100\%$）。
   - 正值：向右推力（$0-100\%$）。
   - 0：不使用侧向引擎。

奖励机制 奖励设计较为复杂，包含以下几个方面：

- 接近着陆区会获得正奖励，远离则获得负奖励。
- 着陆舱移动速度越慢获得的奖励越高。
- 着陆舱倾斜角度偏离水平会获得负奖励。
- 每条着陆腿接触地面获得+10分奖励。
- 每帧使用侧向发动机会扣除0.03分。
- 每帧使用主发动机会扣除0.3分。
- 成功着陆获得额外+100分奖励。
- 坠毁会获得−100分惩罚。

返回目录

终止条件

- 登月舱成功着陆在指定区域。
- 登月舱坠毁。
- 登月舱飞出边界。
- 回合达到最大步数（设定为1000步）。

性能评价（大语言模型提供）

- 因每次构建环境和初始状态不同，得分会有一定差异。
- 最高分约为$200-300$分，
- 新手玩家获得正分数便已说明掌握基本技巧。
- 达到100分以上表示控制水平相当不错。

2.2 策略梯度算法

直接优化策略而非通过价值函数间接优化，参数化策略函数并沿着梯度方向更新。

2.2.1 策略梯度

将策略参数化，搜索策略空间，是同轨策略：

$$\pi(a|s,\theta)=\pi_\theta(a|s)$$

梯度与梯度上升

学习θ使以下指标最大。

- 平均状态价值：

$$\bar{v}_\pi=\sum_{s\in S}d(s)v_\pi(s)=E[v_\pi(S)]$$

其中$d(s)\geqslant 0$为s的权重，$\sum_{s\in S}d(s)=1$，其可由以下方法选取：

  - 均匀分布：$d(s)=\frac{1}{|S|}$。
  - 只关心s_0：$d(s_0)=1,d(s\neq s_0)=0$。
  - 平稳分布：$d_\pi^T P_\pi=d_\pi^T$，根据访问频次赋予概率。

返回目录

- 平均单步奖励：

$$\begin{aligned}\bar{r}_\pi &= \sum_{s\in S} \underbrace{d_\pi(s)}_{\text{平稳分布}} \underbrace{r_\pi(s)}_{\sum_{a\in A}\pi(a|s)} \underbrace{r(s,a)}_{\sum_{r\in R} rp(r|s,a)} = E[r_\pi(S)] \\ &= \lim_{n\to\infty}\frac{1}{n}E[\sum_{k=1}^{n} R_{t+k}] = \lim_{n\to\infty}\frac{1}{n}E[\sum_{k=1}^{n} R_{t+k}|S_t = s_0]\end{aligned}$$

梯度为：

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{s\in S}\eta(s)\sum_{a\in A}\nabla_\theta\pi(a|s,\theta)q_\pi(s,a) \\ &= \sum_{s\in S}\eta(s)\sum_{a\in A}\pi(a|s,\theta)\nabla_\theta\ln\pi(a|s,\theta)q_\pi(s,a) \\ &= E[\nabla_\theta\ln\pi(A|S,\theta)q_\pi(S,A)] \\ &\approx \nabla_\theta\ln\pi(a|s,\theta)q_\pi(s,a)(\text{采样近似})\end{aligned}$$

为确保$\pi(a|s,\theta)>0$，使用softmax函数，$\pi(a|s,\theta)=\frac{e^{h(s,a,\theta)}}{\sum_{a'\in A}e^{h(s,a',\theta)}}$。

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha\nabla_\theta J(\theta) = \theta_t + \alpha E[\nabla_\theta\ln\pi(A|S,\theta_t)q_\pi(S,A)] \\ &= \theta_t + \alpha\underbrace{\nabla_\theta\ln\pi(a_t|s_t,\theta_t)}_{\beta_t=\frac{q_\pi(s_t,a_t)}{\pi(a_t|s_t,\theta_t)}}\underbrace{q_\pi(s_t,a_t)}_{q(s_t,a_t)\text{近似}}(\text{随机梯度})\end{aligned}$$

- $\alpha\beta_t$足够小时，若$\beta_t>0$，则选择(s_t,a_t)的概率增加，且幅度与β_t正相关。
- β_t与$q_\pi(s_t,a_t)$正相关，与$\pi(a_t|s_t,\theta_t)$负相关，倾向于选择高价值动作，探索低概率动作。

优势

- 可以逼近确定性策略，可以逼近任意概率分布，不受$q(s,a)$限制，策略是更简单的函数逼近。
- 策略参数化更容易加入先验知识。
- 在状态空间大时，存储和泛化能力强。

返回目录

2.2.2 REINFORCE（Monte-Carlo Policy Gradient）算法

由Ronald J. Williams于1992年提出。作为最基本的策略梯度算法，用MC估计$q_\pi(s,a)$，使用与θ无关的G_t代替$q_\pi(s_t,a_t)$：

$$\theta_{t+1} \doteq \theta_t + \alpha G_t \nabla_\theta \ln \pi(a_t|s_t,\theta_t)$$

2.3 Actor-Critic算法

结合策略梯度和价值方法。

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \ln \pi(a_t|s_t,\theta_t) q_\pi(s_t,a_t)$$

- 演员（Actor）：策略更新，用于采取行动，对应算法更新。
- 评论家（Critic）：策略评估或价值估计，用于评判策略，对应估计$q_\pi(s,a)$，采用TD方法。

2.3.1 信任区域策略优化（Trust Region Policy Optimization，TRPO）算法

由John Schulman等人于2015年提出，旨在解决策略梯度算法可能导致的过大更新问题，通过约束策略更新步长实现更稳定的训练。

关键技术 在策略更新时维持"信任区域"（trust region），确保新、旧策略间差异不过大，避免灾难性性能下降。

- 限制策略更新：约束新旧策略的KL散度不超过阈值。
- 自然策略梯度：使用Fisher信息矩阵（FIM）计算更新方向，在参数空间均匀更新。

返回目录

替代回报函数

$$\begin{aligned}\eta(\tilde{\pi}) &= \eta(\pi) + \underbrace{E_{s_0,a_0,\cdots\sim\tilde{\pi}}[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)]}_{\text{新旧策略回报差}} \\ &= \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s)\sum_a \tilde{\pi}(a|s)A^\pi(s,a) \\ L_\pi(\tilde{\pi}) &= \eta(\pi) + \sum_s \rho_\pi(s)\sum_a \tilde{\pi}(a|s)A^\pi(s,a)(\text{忽略状态分布变化}) \\ &= \eta(\pi) + E_{s\sim\rho_{\theta_{old}},a\sim\pi_{\theta_{old}}}[\frac{\tilde{\pi}_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}A_{\theta_{old}}(s,a)](\text{重要性采样动作分布})\end{aligned}$$

有$L_{\pi_{\theta_{old}}}(\pi_{\theta_{old}}) = \eta(\pi_{\theta_{old}}), \nabla_\theta L_{\pi_{\theta_{old}}}(\pi_\theta)\big|_{\theta=\theta_{old}} = \nabla_\theta\eta(\pi_\theta)\big|_{\theta=\theta_{old}}$。
令$\alpha = D_{TV}^{max}(\pi,\tilde{\pi}), \epsilon = \max_{s,a}|A_\pi(s,a)|$，惩罚因子$C = \frac{2\epsilon\gamma}{(1-\gamma)^2}$，则有：

$$\eta(\tilde{\pi}) \geqslant L_\pi(\tilde{\pi}) - \frac{4\epsilon}{(1-\gamma)^2}\alpha^2 \quad \eta(\tilde{\pi}) \geqslant L_\pi(\tilde{\pi}) - CD_{KL}^{max}(\pi,\tilde{\pi})$$

优化：共轭梯度搜索 问题转化为：

$$\begin{aligned}&\max_\theta[L_{\theta_{old}}(\theta) - CD_{KL}^{max}(\theta_{old},\theta)] \\ \xRightarrow{\text{迭代步长较小}}\ &\max_\theta E_{s\sim\rho_{\theta_{old}},a\sim\pi_{\theta_{old}}}[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}A_{\theta_{old}}(s,a)], \quad D_{KL}^{max}(\theta_{old},\theta) \leqslant \delta \\ \xRightarrow[\text{平均KL散度}]{\text{策略状态空间分布}}\ &\max_\theta E_{s\sim\pi_{\theta_{old}},a\sim\pi_{\theta_{old}}}[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}A_{\theta_{old}}(s,a)], \quad E_{s\sim\pi_{\theta_{old}}}\{D_{KL}[\pi_{\theta_{old}}(\cdot|s)\pi_\theta(\cdot|s)]\} \leqslant \delta \\ \xRightarrow[\text{约束二次逼近}]{\text{目标线性逼近}}\ &\max_\theta[\nabla_\theta L_{\theta_{old}}(\theta)\big|_{\theta=\theta_{old}}\cdot(\theta-\theta_{old})], \quad \frac{1}{2}(\theta_{old}-\theta)^T\underbrace{A(\theta_{old})}_{\text{Fisher信息矩阵}}(\theta_{old}-\theta) \leqslant \delta\end{aligned}$$

计算复杂性

- 计算FIM：需要计算策略的二阶导数。
- 求解线性方程系统：FIM通常是高维矩阵，$Ax = b$计算复杂。
- 线搜索：为满足KL约束需多次策略评估。

2.3.2 近端策略优化（*Proximal Policy Optimization*，*PPO*）算法

由OpenAI于2017年提出，结合TRPO的稳定性和简单实现的优势，是最流行的强化学习算法之一。

返回目录

关键技术

- 裁剪目标函数：限制新旧策略间差异，避免过大策略更新。

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta) \cdot A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t)]$$

  其中，A_t是优势函数估计，ϵ是裁剪超参数，限制范围为$[1-\epsilon, 1+\epsilon]$。

- 多回合更新：从相同轨迹数据中多次更新，提高样本利用效率。
- 重要比率$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$。
- GAE（Generalized Advantage Estimation，广义优势估计）：

  由基本优势函数$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$演变得到GAE优势：

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

  其中，$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$。$\lambda$趋于0时为纯TD估计，方差低但偏差高；$\lambda$趋于1时为蒙特卡洛估计，方差高但偏差低；一般取接近1的值进行平衡。

目标 包含值函数损失和熵正则化项：

$$L^{TOTAL}(\theta) = L^{CLIP}(\theta) - c_1 \cdot L^{VF}(\theta) + c_2 \cdot S\pi_\theta$$

其中，L^{VF}是值函数的均方误差损失，S是策略熵函数，c_1, c_2是系数超参数。

优势

- 实现简单：不需要计算二阶导数。
- 样本利用效率高：通过多回合更新提高数据效率。
- 稳定性好：裁剪机制限制了策略更新步长。
- 广泛适用：可用于连续和离散动作空间，在多环境性能出色。

2.3.3 深度确定性策略梯度（*Deep Deterministic Policy Gradient*，*DDPG*）算法

由Google DeepMind于2015年提出，结合DQN和确定性策略梯度，属于off-policy算法。

$$\nabla_\theta J(\theta) = \sum_{s \in S} \underbrace{\rho_\mu(s)}_{\text{状态分布}} \nabla_\theta \mu(s) [\nabla_a q_\mu(s, a)] \Big|_{a=\mu(s)} = E_{S \sim \rho_\mu} \{ \nabla_\theta \mu(S) [\nabla_a q_\mu(S, a)] \Big|_{a=\mu(S)} \}$$

返回目录

核心思想

- 采用Actor和Critic两种网络，前者输入状态，输出确定性动作；后者输入状态和动作，输出Q值。由于目标网络（Target Network）的存在，一共是四个网络。
- 采用经验回放（Replay Buffer）提升训练稳定性。
- 在动作选择时依赖OU噪声进行探索：
- OU噪声由以下随机微分方程定义：

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

其中：X_t时噪声值；θ是回归速度参数，控制噪声回归到均值的速度；μ是长期均值；σ是噪声强度参数；dW_t是维纳过程的白噪声。

离散化后的更新公式为：

$$X_{t+1} = X_t + \theta(\mu - X_t) + \sigma\epsilon_t$$

其中$\epsilon_t \sim \mathcal{N}(0,1)$是标准正态分布噪声。

优势

- 能处理高维、连续动作空间问题。
- 训练稳定，收敛速度快。

2.4 差异比较

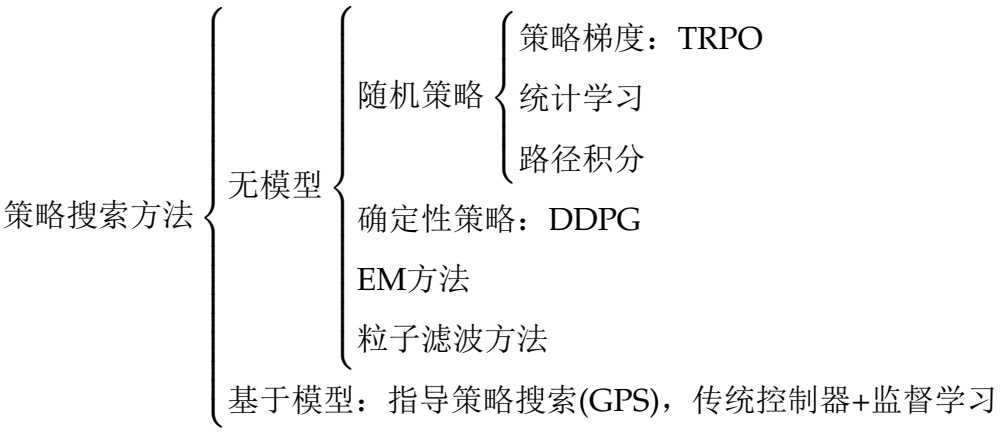
- REINFORCE：最基本的蒙特卡洛策略梯度算法，直接利用采样轨迹估计策略梯度，更新全部参数，方差较大，收敛慢。
- PPO：在REINFORCE基础上引入裁剪目标函数，限制策略更新幅度，结合优势函数（GAE）和基线（值函数网络），多回合更新，显著提升了训练的稳定性和样本利用效率。
- DDPG：采用确定性策略，适用于高维连续动作空间。引入经验回放和目标网络，训练更稳定，收敛速度快，适合复杂控制任务。

返回目录

表 1: 算法对环境的适用性

算法	离散环境	连续环境
REINFORCE	√	x
PPO	√	√
DDPG	x	√

- REINFORCE在修改网络输出为正态分布后可以应用于连续环境，但是由于高方差和探索效率低，其收敛极为困难。
- DDPG是为确定性连续策略设计的，在处理离散环境时，由于无法计算动作的微分，策略网络传播少了重要部分。



3 关键代码解析

由于代码较多，不作展示，请见附件，以下就关键点进行说明。

3.1 REINFORCE

在基础REINFORCE代码上进行了一定改进。

采样

返回目录

- 为避免智能体陷入训练停滞，设置了动态时间惩罚——超过200步后每步有较小惩罚。该惩罚只作用于训练中。
- 定义了采样一条轨迹的函数，是采样多条轨迹的基础。

网络

- 策略网络采用三个全连接层，激活函数为ReLU。
- 策略网络层正交初始化权重，常数初始化偏置（默认为0）。
- 通过Categorical分布输出离散动作空间上的概率分布。
- 计算概率时采用对数。

REINFORCE

- 熵正则化，鼓励探索。
- 多轮SGD更新，每次采样后进行多次小批量训练。
- 标准化回报，减少方差。
- 随机打乱训练数据，提高泛化性能。
- 指数衰减学习率。
- 梯度裁剪，防止梯度爆炸，提高训练稳定性。

3.2 PPO

采样

- 采用GAE优势，减少估计方差。
- 定义了采样一条轨迹的函数，是采样多条轨迹的基础。

返回目录

网络

- Actor、Critic网络采用三个全连接层，激活函数为Tanh。
- 网络层采用正交初始化权重，常数初始化偏置（默认为0）。
- 通过Categorical分布输出离散动作空间上的概率分布。
- 计算概率时采用对数。

PPO

- 裁剪目标函数，限制策略更新幅度，避免过大策略变化。
- 使用重要性采样比率衡量新旧策略差异。
- 熵正则化，鼓励探索。
- KL散度监控，当超过阈值时提前停止内层更新。
- 分别优化Actor和Critic网络，采用不同的学习率。
- 多轮SGD更新，每次采样后进行多次小批量训练。
- 标准化优势函数，减少方差。
- 随机打乱训练数据，提高泛化性能。
- StepLR学习率调度器，指数衰减。
- 梯度裁剪，防止梯度爆炸，提高训练稳定性。
- 早停机制，当性能无改进时提前终止训练。

3.3 DDPG

经验回放

- 使用经验回放缓冲区存储转移元组。
- 设置起始步数，在收集足够经验前使用随机动作。

返回目录

网络

- Actor、Critic网络采用三个全连接层，Actor网络激活函数为Tanh，Critic网络直接输出Q值估计。
- 网络层采用正交初始化权重，常数初始化偏置（默认为o）。
- 软更新目标网络跟踪主网络。

DDPG

- 随机采样训练批次，减少样本相关性。
- 实现OU噪声进行连续动作空间探索，并裁剪动作，确保在合法范围内。
- 分别优化Actor和Critic网络，采用不同的学习率。
- 指数衰减学习率。
- 多轮SGD更新，每次采样后进行多次小批量训练。

返回目录

4 实验结果与分析

4.1 参数设置

表 2: 参数设置

参数名称	REINFORCE	PPO	DDPG
训练环境	离散	离散	连续
隐藏层大小	[128, 128]	[64, 64]	[256, 256]
策略学习率	0.0003（指数衰减）	1e-4（StepLR）	3e-4（指数衰减）
价值学习率	–	5e-4（StepLR）	1e-3（指数衰减）
批大小（SGD）	256	64	64
每轮采样轨迹数	10	8	–
训练轮数	10000	1000（早停）	1000
每轮SGD迭代次数	5	10	–
最大梯度裁剪	0.5	0.3	–
熵正则化系数	0.01	0.01	–
PPO裁剪比例	–	0.1	–
KL散度阈值	–	0.02	–
GAE参数λ	–	0.95	–
折扣因子γ	0.99	0.99	0.99
模型保存频率	100	100	100
早停容忍轮数	–	50	–
软更新参数τ	–	–	0.005
噪声标准差	–	–	0.1
经验回放缓冲区大小	–	–	100000
学习率衰减率	–	–	0.995
最小学习率	–	–	1e-6
随机探索步数	–	–	10000
更新频率	–	–	50

返回目录

表 3: 实验结果

指标	REINFORCE	PPO	DDPG
训练环境	离散	离散	连续
耗时（s）	3534.85	296.74	6317.80
轮数	10000	1000（早停于300前）	1000
平均奖励	211.88	175.08	167.60（有极端情况）
最高奖励	248.83	209.90	282.37
平均步数	335.70	773.30	456.00

4.2 实验结果展示

- 平均步数：由于REINFORCE有时间惩罚项，模型在充足训练后以较少步数完成任务，比PPO要快一倍还多。DDPG经过较为充分的训练，平均步数接近REINFORCE。
- 奖励：在平均奖励和最高奖励上，REINFORCE都要高于PPO35分左右，这与步数差距是吻合的。需要注意，这里并不是指时间惩罚项，而是指发动机使用的燃料消耗惩罚，因为前者并不会在测试时使用。DDPG的平均奖励最低，但是其最高奖励却最高，这是因为在测试的部分回合中，出现少数零值乃至负值结果。如果抛开这些极端情况，在绝大部分回合中，DDPG能取得比REINFORCE还要高的平均奖励。
- 耗时：尽管单轮训练REINFORCE要比PPO和DDPG快得多，PPO快速早停使其在极少的轮数终止，耗时极少。DDPG由于需要经验回放和目标网络更新，耗时最长。

另外，本人简单尝试该游戏，得分为0分左右，符合基本掌握规则的水平。也邀请其他同学进行了测试，刚开始得分在−50分左右，之后也达到0分左右。然而，模型测试时和实际游戏时，可分辨帧数是不同的，这也是入门玩家分数较低的原因。

4.3 实验过程分析

4.3.1 REINFORCE

下图2 on the next page的回报曲线展示了REINFORCE在训练过程中回报的波动情况。在初期（前2000轮）训练时，波动较大，之后幅度变小，结果不断优化。后期有一个尖峰波动，之后效果略微下降。

返回目录

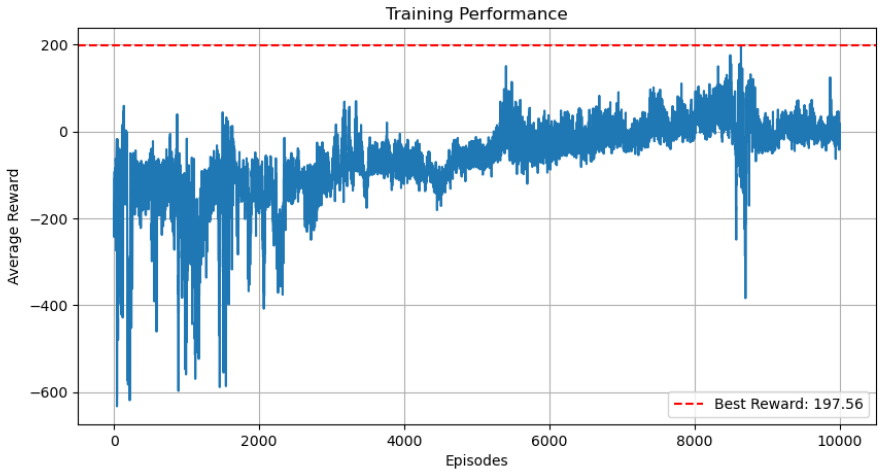


图 2: REINFORCE回报曲线

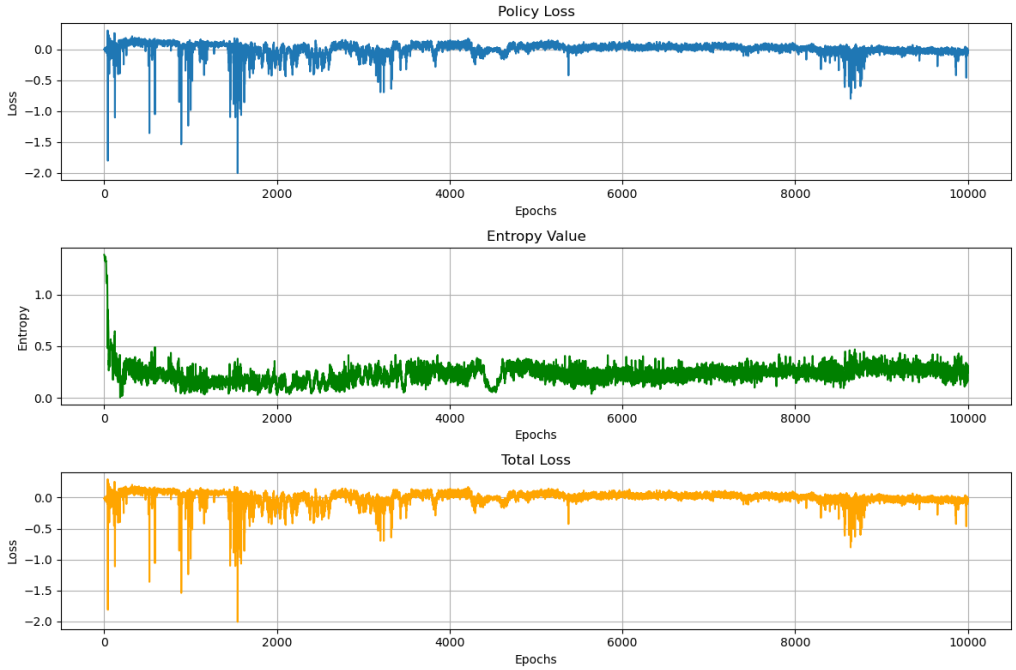


图 3: REINFORCE损失曲线

上图3的REINFORCE损失曲线分为三个子图，由上到下依次是策略损失、熵值和总损失。策略损失是一个对数值，描述了策略优化的空间。熵值是探索性鼓励因子，源于策略网络的动作输出。二者综合组成总损失。

返回目录

代码中策略损失是一个负值$\text{loss}_\text{p}\text{i} = -(\text{logp} * \text{value}).\text{mean}()$，熵值是一个正值$\text{self.actor}(\text{obs}, \text{act})$，在综合时求差$\text{loss} = \text{loss}_\text{p}\text{i} - 0.01 * \text{loss}_\text{e}\text{ntropy}$，因此三者都以趋于零为收敛。

观察变化曲线，与回报曲线相吻合，在初期波动较大，之后逐渐收敛，后期有一个小的尖峰波动。

4.3.2 PPO

下图4的回报曲线展示了PPO在训练过程中回报的波动情况。回报的波动幅度一直较小，整体得分呈明显上升趋势。由于早停机制的引入，模型在300轮前就停止了训练。

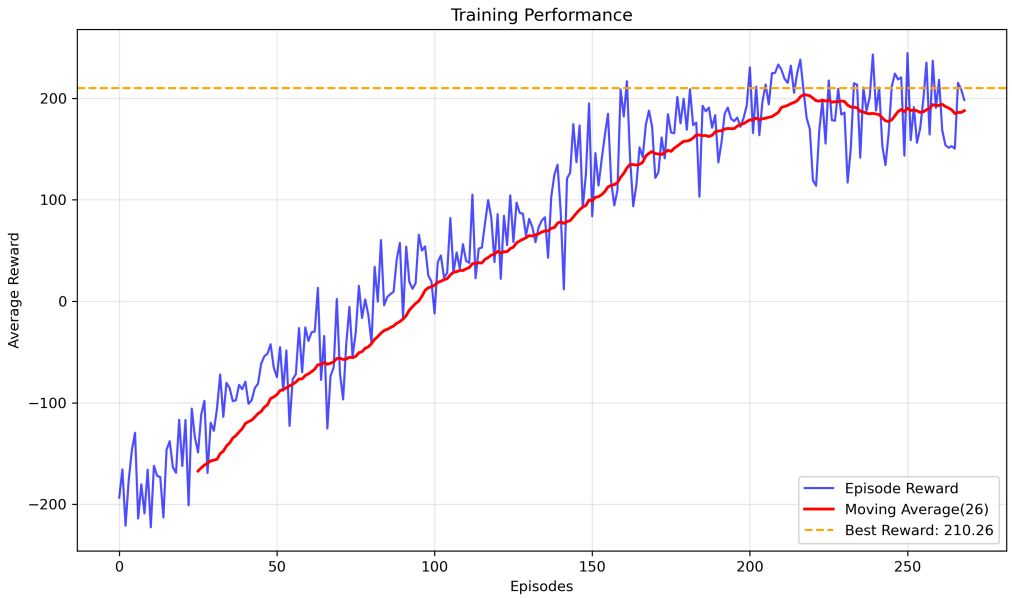


图 4: PPO回报曲线

返回目录

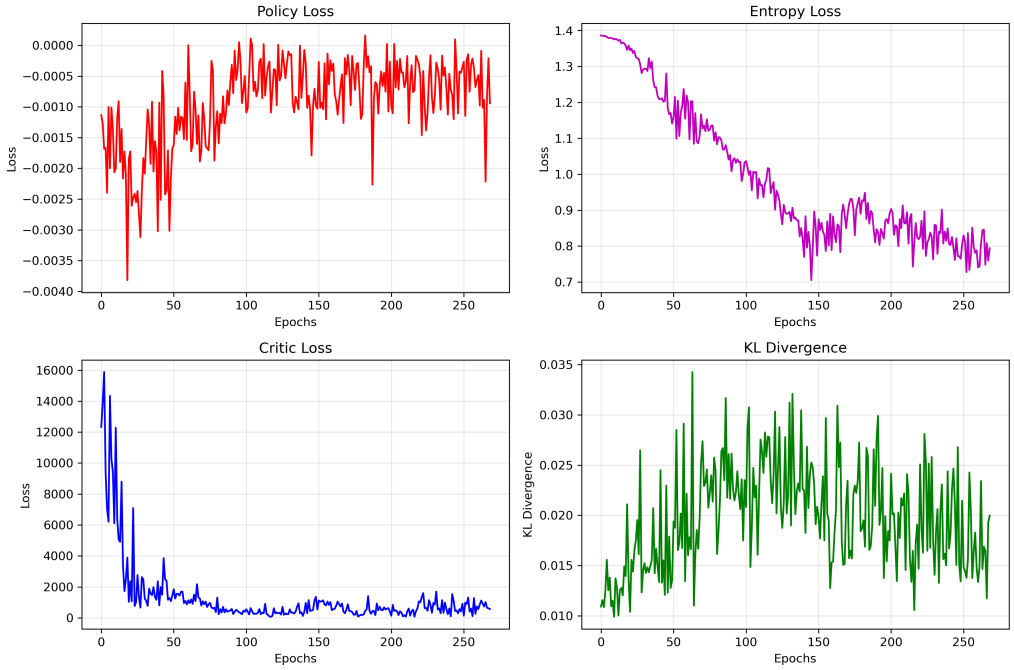


图 5: PPO损失曲线

上图5的PPO损失曲线分为四个子图，由左上到右下依次是策略损失、熵值、评价损失和KL散度。评价损失衡量评论家网络预测价值的准确性，KL散度则衡量新旧策略间差异。

代码中策略损失是一个裁剪后的负值$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)\cdot A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\cdot A_t)]$，熵值是一个正值$entropy = dist.entropy()$，评价损失是未裁剪的正值$L^{VF}(\theta) = Et[(V\theta(s_t) - V_t^{target})^2]$，而KL散度是近似值$ApproxKL = E[|\log \pi_{\theta_{old}}(a|s) - \log \pi_\theta(a|s)|]$。前三者都以趋于零为收敛。

观察变化曲线，与回报曲线相吻合，在初期波动较大，之后快速收敛，这表现了探索阶段向优化阶段的转变。观察KL散度的变化曲线，呈现略微凸起的形状，这对应了初入优化期时的高效开发，在找到优质策略后变化量逐渐减少。

4.3.3 DDPG

下图6 on the next page的回报曲线展示了DDPG在训练过程中回报的波动情况。回报的波动幅度较大，上升趋势不明显，峰值较低，且后期有下降趋势。然而，训练过程中的测试结果的平均得分要高于训练结果，如最终保留的模型数据在训练中的测试平均分为280。

返回目录

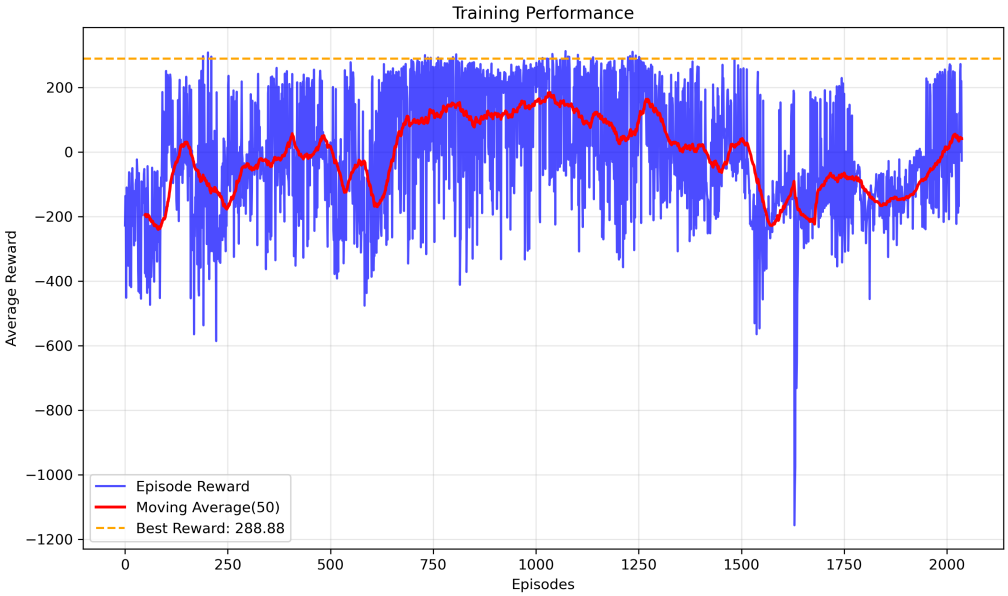


图 6: DDPG回报曲线

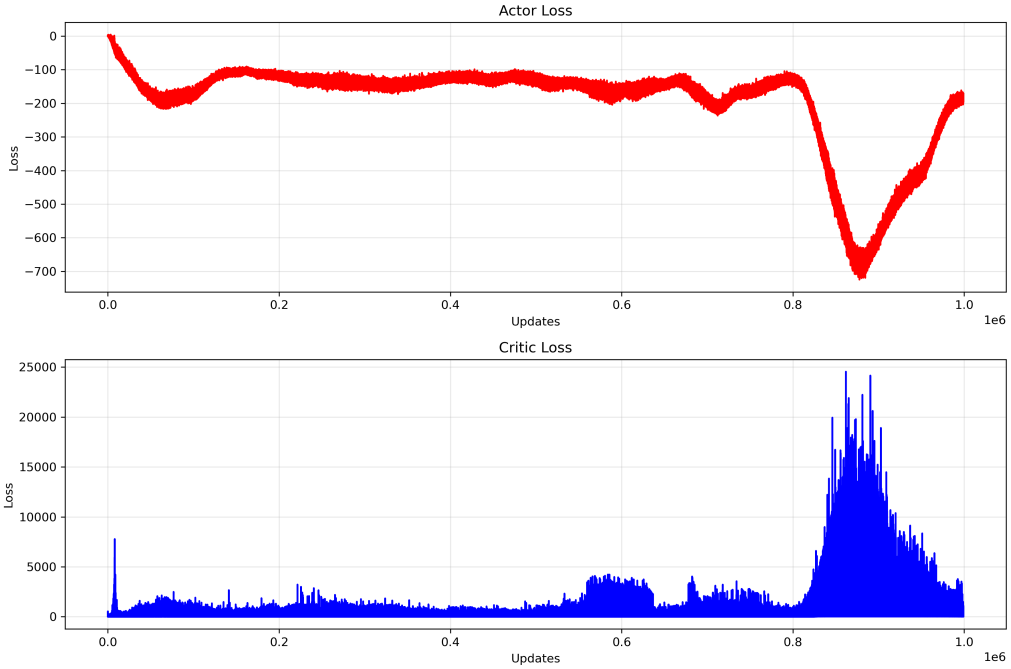


图 7: DDPG损失曲线

上图7的DDPG损失曲线分为两个子图，上图是表示策略网络性能的演员损失，下图是表示价值网络估计精度的评论家损失。

返回目录

演员损失是负值$-self.critic(state, self.actor(state)).mean()$，评论家损失是正值$F.mse_loss(current_q, target_q)$，二者都以趋于0为优。

观察变化曲线，与回报曲线相吻合，波动一直较大（网格跨度较大），没有收敛趋势，后期还有一个尖峰波动。经分析，收敛性较差可能源于：

- OU噪声参数设置不当。
- Actor和Critic网络学习速度不匹配。
- 经验回放缓冲区大小设置不当。
- 目标网络更新频率不当。

4.3.4 比较分析

由于未控制参数一致，并且训练的环境不同，其实无法直接比较，仅就结果做简要分析。

- 收敛速度：PPO由于有早停机制，可以给出明确的收敛时间，大概是300轮0.008h；而REINFORCE则由于大幅振荡，无法适用早停机制，收敛时间大概是5000轮0.5h。虽然没有控制公共参数，但是也可以说明PPO的收敛速度远高于REINFORCE。DDPG算法虽然有较好结果（保存的最优结果为300轮0.6h左右），但是一直没有收敛。
- 训练过程稳定性：比较回报曲线，REINFORCE的震荡要远大于PPO。由于两个图的比例尺不一致，这一点可能不太明显。PPO的滑动平均曲线标定了较小的波动范围，而REINFORCE多次经历触底反弹的大幅波动。DDPG的训练过程极不稳定，振荡幅度高出前两者一个数量级，从回报上来看，是相邻轮交替出现极好极坏的结果。
- 收敛结果：结合上一部分分析，当前REINFORCE的收敛结果要优于PPO，其在较少步数下稳定降落，节省了燃料。DDPG的结果虽然没有收敛，但是也能在绝大部分情况下极好地完成任务。
- 超参数敏感性（大模型提供）：DDPG > REINFORCE > PPO。DDPG对学习率、噪声参数、网络结构等超参数较为敏感；REINFORCE对学习率敏感；PPO相对鲁棒。

返回目录

5 实验思考

- 在尝试为REINFORCE添加早停机制时，即使200轮的阈值仍会在2000轮内终止，得到极差的结果，故放弃早停，设置足够的轮数。
- 最初也为PPO设计了时间惩罚项，但是3000轮仍无法突破100分，且长期处于负分。观察智能体策略，发现其一直保持在窗口顶部不下降，消耗燃料直到最大时间步。之后尝试添加高度惩罚，仍然无法改进。最后选择回归游戏自身的奖励机制。
- 在对PPO进行无早停实验时，无法在当前峰值上继续提升，会大幅跳水，在1000轮之内无法突破当前峰值，这说明其可能在较长时间内止步于局部最优解。
- 在尝试使用PPO训练连续环境时，修改动作分布，屡次调整训练餐宿，仍是效果极差，故未做展示。
- 由于训练环境的数据需要在CPU上传递，在GPU上训练会大幅增加传递耗时，所有算法都统一在CPU上训练。

6 实验总结

本次实验尝试了REINFORCE算法、PPO算法和DDPG算法。对比实验结果，可以发现，PPO的收敛速度和训练过程中的稳定性都要远优于REINFORCE。而DDPG对于连续环境也是有效的算法。

返回目录