Reinforcement Learning Final Project

叶增渝 123033910090

2023 年 12 月 26 日

一、 基于值的强化学习方法

1.1 游戏环境总览

我们需要从 gymnasium 库的 Atari 2600 经典游戏环境中选择 VideoPinball-ramNoFrameskip-v4、BreakoutNoFrameskip-v4、PongNoFrameskip-v4、BoxingNoFrameskip-v4 其中一个作为游戏环境，完成基于值的强化学习智能体的训练。

这些选择的游戏环境具有共同的特性：没有任何的帧跳过操作，智能体观察得到的环境即为做出当前动作得到的直接结果。智能体在这样的游戏环境中可以控制每一帧进行的动作，相对难度最低。但是这些游戏之间依然存在较大差异：1）VideoPinball-ramNoFrameskip-v4 为经典三维弹球游戏，就算不做出任何动作，一次游戏也会有几千步，得到上千的奖励，导致训练一个回合时间过长，该游戏的特点是环境奖励与可执行动作没有强关联，导致智能体难以训练；2）BreakoutNoFrameskip-v4 为打砖块游戏，需要主动移动板子以反弹小球击破砖块得分，该游戏的特点是初始步骤难度较高，导致智能体容易在初始反复尝试，难以找到较好的回合参数进行参数迭代；3）PongNoFrameskip-v4 与 BoxingNoFrameskip-v4 均为与 AI 玩家进行对打的游戏，由于需要学习 AI 玩家的动作规律，所以推断 AI 行动逻辑阶段需要多回合尝试，导致前期回合较短，智能体难以学习到有用数据。特别的，VideoPinball-ramNoFrameskip-v4 的游戏环境默认 obs_type 为 ram，使用 128 个 uint8 表示游戏当前状态。

我们使用与上述 4 个指定环境同名的 gymnasium=0.29.1 中对应的 atari 游戏环境，且同时规定所有 obs_type='ram'。我同时对 4 种游戏进行智能体学习并进行结果展示，基于不同游戏，对应的默认模型超参数也会不同。

1.2 方法概览

我们在这里使用的强化学习方法为 RainbowDQN，其由 DeepMind 提出，融合了 DoubleDQN、DuelingDQN、NoiseLayer、Prioritized Experience Replay、Multi-Step Learning、Distributional RL 等技术，具有强大的学习能力，下面我们会详细解析 RainbowDQN 使用到的每一个思想。

在最基础的 DQN 模型基础上，人们观测到 DQN 对 Q 值的预测往往会过高，导致模型选择过高估值的动作，导致结果出现偏差。因此 DoubleDQN 将 Q 值估计从 (1) 转为 (2)，从而防止模型进行过高的 Q 值预测。

$$Q(s_t, a_t) \longleftarrow r_t + \max_a Q(s_{t+1}, a) \tag{1}$$

$$Q(s_t, a_t) \longleftrightarrow r_t + Q^{'}(s_{t+1}, arg\,\underset{a}{max}\, Q(s_{t+1}, a)) \tag{2}$$

此外，RainbowDQN 还使用了 DuelingDQN 的支路思想来适应不同环境情况，将 Q 值分为价值函数部分与优势函数部分，并且为辨别 2 个网络各自的作用，还要对优势函数做出一点修改，最终如 (3)。

$$Q(s, a\,\theta; \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|A|}\sum_{a'} A(s, a'; \theta, \alpha)) \tag{3}$$

RainbowDQN 也在 ReplayBuffer 上下了功夫，不同于普通的随机采样，RainbowDQN 使用了 Prioritized Experience Replay。对于在之前训练过程中拥有更高 TD error 的数据，它会有更大的采样概率。

为了增强模型的探索能力，NoisyNet 也是 RainbowDQN 结合的一个部分，它采取其中的 Noise on Parameters 的方法，将优势函数部分的全连接层改为噪声层来添加 Noise，所以整体上看，模型的预测转变如 (4)。

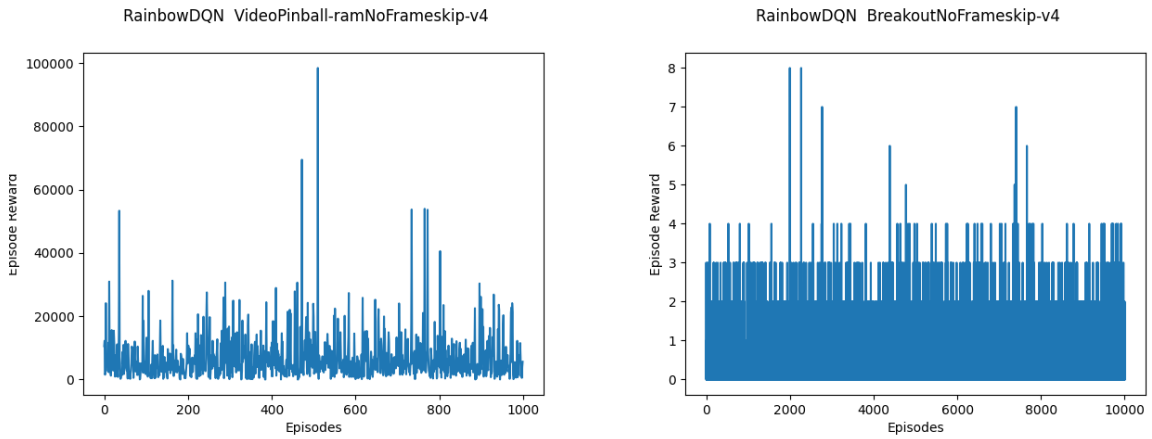
$$a = arg\,\underset{a}{max}\, \tilde{Q}(s, a)) \tag{4}$$

DistributionalDQN 也是 RainbowDQN 的借鉴对象，它将“Q 值”拓展到了“Q 分布”，对每一个动作不再只输出一个 logits，而是一个多值离散分布 (多个 bin)，它基于 Bellman 最优性算子，得到一个价值分布。

Multi-Step Learning 的采用则主要是为了提高学习能力。原始 DQN 每一步学习以此导致模型参数更新次数过多，所以采用 k 步累积来加速学习，同时这样的方法也能在强化学习前期得到更准确的 Q 值估计。

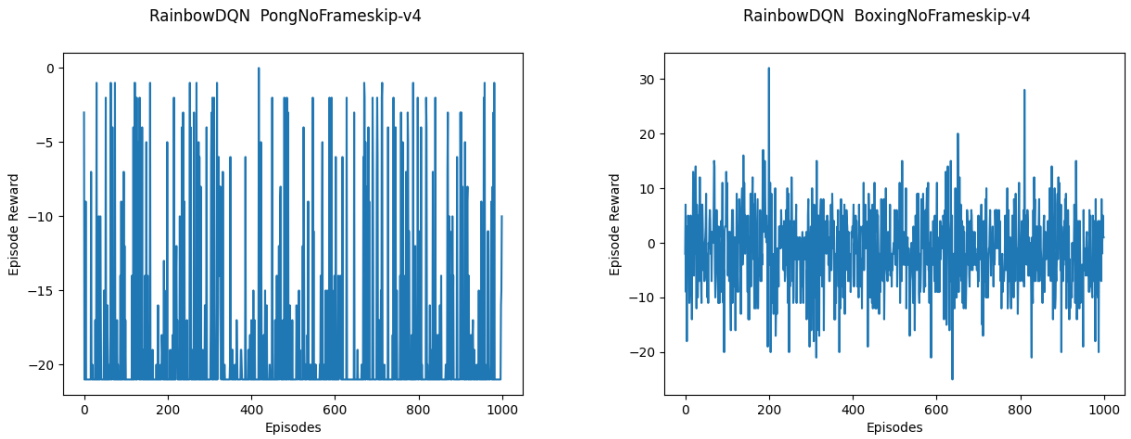
综上，我在任务 1 中使用了 RainbowDQN 的网络结构构造游戏智能体完成游戏，不同的游戏设定的参数不同，可以通过源文件 RainbowDQN.py 中的 argparser 查看可设定参数。

1.3 模型表现展示



(a) VideoPinball (b) Breakout

图 1: 基于值的强化学习奖励训练曲线



(a) Pong (b) Boxing

图 2: 基于值的强化学习奖励训练曲线 (续)

1.4 结果分析

首先，由于各个游戏的一个 episode 的所需时间均不同，为防止部分智能体训练时间过长，我对 PongNoFrameskip-v4 与 VideoPinball-ramNoFrameskip-v4 采取 1000 个训练 episode，对 BreakoutNoFrameskip-v4 与 BoxingNoFrameskip-v4 采取 10000 个训练 episodes。

从奖励曲线变化我们不难看出，智能体的训练不是很成功，只能和初始状态基本持平，其中 PongNoFrameskip-v4 为训练相对较为成功的智能体，虽然在后期依然有较大的波动，但是基本能有一半的尝试能够达到 0 的奖励，而其余模型虽然也可能获得较好的奖励，但基本都是偶然事件，没有被智能体完全学习到相关特征，最终与随机策略没有明显的优势。

经过社区讨论与环境奖励分析，我认为主要有两个原因：首先，这些 atari 游戏环境的奖励函数与动作的关联性不够强烈，往往与经典游戏中的得分为相同的设置，例如 VideoPinball-ramNoFrameskip-v4 中，如果小球不碰到特定位置就无法得分，所以此类位置上返回的每一步奖励均为 0，那么智能体就无法辨别当前是否有危险需要做特定动作，也认识不到动作的必要性，最终也就效果很差。其次，游戏本身的难度较高，比如在 BreakoutNoFrameskip-v4 中，如果没有办法及时做出正确动作，一个 episode 马上就会结束，较大的难度会导致智能体前期无法学习到有用策略。

主要是上述 2 个原因导致了该任务训练效果不佳，可能需要进一步调整超参数与训练方法，才能拟合出较好的游戏智能体。

二、 基于策略的强化学习方法

2.1 任务总览

我们需要从 gymnasium 库的 mujoco 物理模拟环境中选择 Hopper-v2、Humanoid-v2、HalfCheetah-v2、Ant-v2 其中一个作为模拟环境，完成基于策略的强化学习智能体的训练。

这些选择的模拟环境具有共同的特性：它们都希望对应物理环境中的模拟物体能够保持在好的状态，一旦进入不好的状态 (如倒地)，该次尝试就会结束。智能体在这样的模拟环境中可以控制每一帧相关关节进行的动作方向与速度，不同的模拟环境中可动的关节、整体形状、观察空间均不同，但是数值范围基本相似。其中 1) Hopper-v2 为形状类似扫帚的几何物体，可进行类似扭动的动作，需要维持竖直不倒地；2) Humanoid-v2 为类似火柴人的几何物体，需要维持站立不倒地；3) HalfCheetah-v2 为类似小狗的几何物体，需要维持四脚着地，头悬空；4)Ant-v2 类似一个四角蜘蛛，需要维持身体悬空支撑。虽然观察空间、动作空间可能数组数量不一样，但是全为 (-inf, inf) 上的浮点数据，4 个环境可以完美迁移，使用相同的超参数。

我们使用与上述 4 个指定环境同名的 gymnasium=0.29.1 中对应 mujoco 游戏环境，由于 mujoco 中 v3 以下的版本需要安装 mujoco-py，与 gymnasium 冲突较严重，所以我选择使用上述 4 个环境的 v4 版本，并使用默认参数，从而达到与 v2 相同的模拟环境 (最大的好处在于直接 pip 安装而不需要额外的库，无需解决版本冲突问题)。我同时对 4 种游戏进行智能体学习并进行结果展示，基于不同游戏，对应的默认模型超参数也会不同。

2.2 方法概览

我们在这里使用的强化学习方法为 DDPG，它结合了深度学习和强化学习，主要用于解决连续动作空间的问题。它直接学习一个策略，该策略可以映射观察到的状态到具体的动作，它的目标是最大化价值 V，利用梯度上升法最大化 Q 函数，且学习确定性策略 (对于给定的状态，策略总是产生相同的动作)。DDPG 使用策略网络中经典的 Actor-Critic 架构，Actor 用于生成动作，而 Critic 用于评估这些动作的价值，这样有助于学习的稳定。

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

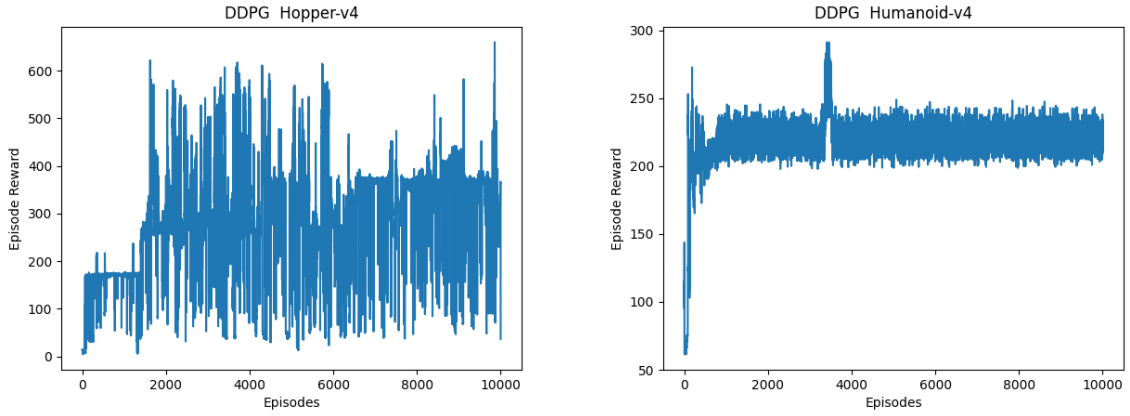
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

除此之外，DDPG 也含有 DQN 的一些经典架构，包括经验回放缓冲池与目标网络记住，用于提高学习的稳定性和避免自我参照问题。

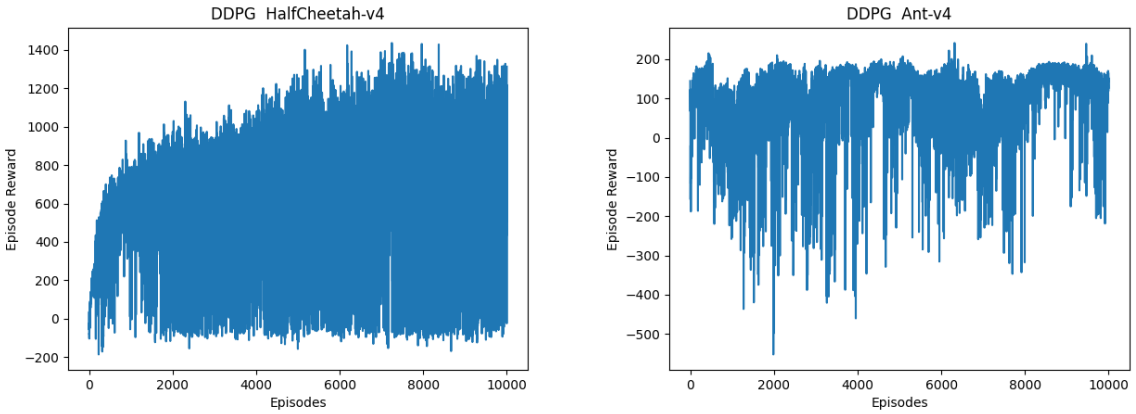
总的来说，DDPG 是一种能够处理复杂的、连续动作空间的强化学习问题，在上述的复杂连续环境中能够有效学习策略，非常适合上述的 4 个任务。

2.3 模型表现展示



(a) Hopper (b) Humanoid

图 3: 基于策略的强化学习奖励训练曲线



(a) HalfCheetah (b) Ant

图 4: 基于策略的强化学习奖励训练曲线 (续)

2.4 结果分析

由于各个模拟环境相似,所以均采取 10000 个训练 episode,并使用相同的超参数。

对于 Hopper,奖励首先从 0 快速抬升到 160 左右,然后在 0 1500 个 episode 稳定,接着在 1500 6500 个 episode 时奖励在 250 左右上下波动,然后在 6500 10000 个 episode 时奖励在 350 的均值上下波动;由于在训练中策略选取存在随机性产生的波动,实际测试时智能体的 episode 奖励会稳定在 350 左右,训练效果较好但不够稳定;对于 Humanoid,奖励在前几十个 episode 就稳定抬升,并在之后持续稳定在 200 230 左右,虽然在 3500 个 episode 存在抬升到近 300 的奖励,但是智能体最终没有抓取到相关特征来获得更好的表现;对于 HalfCheetah,可以看到智能体的奖励始终波动非常明显,但是总体能达到的最高奖励在不断升高至近 1400,还是在不断攀升,在训练中策略选取存在随机性产生的波动,实际测试时智能体的 episode 奖励会稳定更加的稳定;对于 Ant,奖励在前几十个 episode 就稳定抬升,从初始的负值迅速攀升至 100 200 之间,虽然后续可能存在更大的负值,但奖励最终还是稳定在了 150 左右,可以观察到智能体在第 6000 8000 个 episode 时均值有下降,但是智能体成功矫正了错误,重新提升了 episode 奖励。

与基于值的强化学习任务相比,基于策略的强化学习任务的观察空间与奖励函数设计更加合理,因此能够取得较好的学习效果,我们在 4 个模拟环境下均取得较好的成绩。

三、 附录

如想要正确运行相关源代码，请查看压缩包中的 RL 文件夹，以 RL 文件夹为根目录，根据其中的 README.md 文件进行环境配置与运行测试，以下会粘贴 README.md 中的相关要求。

环境安装 (Linux 环境、RL 文件夹为根目录)：

```
conda create -n RL python==3.11
conda activate RL
pip install gymnasium
pip gymnasium[accept-rom-license]
pip gymnasium[atari]
pip gymnasium[mujoco]
pip install matplotlib
pip3 install torch torchvision torchaudio
pip install tensorboard
pip install pillow
```

帮助命令：

```
python RainbowDQN.py --help
python DDPG.py --help
```

运行命令：

```
# RainbowDQN的可选环境为 'VideoPinball-ramNoFrameskip-v4',
# 'BreakoutNoFrameskip-v4', 'PongNoFrameskip-v4', 'BoxingNoFrameskip-v4'
python RainbowDQN.py --env_name VideoPinball-ramNoFrameskip-v4

# DDPG的可选环境为 'Hopper-v4', 'Humanoid-v4', 'HalfCheetah-v4', 'Ant-v4'
python DDPG.py --env_name Hopper-v4
```

生成文件保存位置：最终模型会保存在 /models 文件夹下，最终“奖励-episode 曲线”保存在根目录下，图像名为'$model_name$_$task_name$.png'