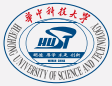


HUAZHONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

AI 派第二轮测试答辩（RL 方向）

报告人：刘明宇

华中科技大学，船舶与海洋工程

2025 年 9 月 29 日

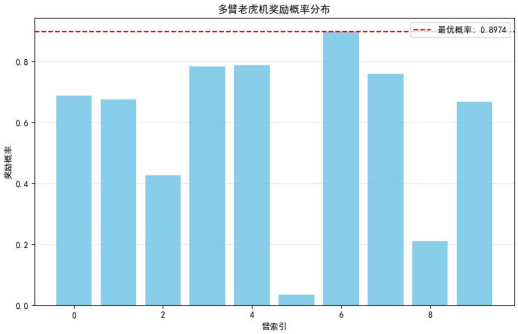
大纲

1. 任务一：Bandit
2. 任务二：Gym-taxi
3. 任务二补充：Alien-Atari-v5
4. 任务三：论文阅读
5. 任务四：Pendulum-v1

大纲

问题介绍

figure1:每个bandit中奖的概率



目标：1.获取尽可能高的分数
2.使用Bellman方程解决上述问题

动态规划：用Bellman方程更新动态规划本质上就是利用 贝尔曼方程（Bellman Equation）来递归分解最优值函数的计算问题。它要求环境是已知的马尔可夫决策过程（MDP）

价值函数满足一个递推关系——就是 **贝尔曼方程。**

- 对状态价值函数：

$$V^{\pi}(s)=\sum_{a}\pi(a|s)\sum_{s'}P(s'|s,a)\Big[R(s,a)+\gamma V^{\pi}(s')\Big]$$

- 对动作价值函数：

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

在Bandit环境中：

$$V(s)=\max_{a}Q(s,a)$$

$$V(s)=\max_{a}\left(\mathbb{E}[R(a)]+\gamma V(s)\right)$$

$$V(s)=\frac{\max_{a}\mathbb{E}[R(a)]}{1-\gamma}$$

$$\pi^{*}(s)=\arg\max_{a}\mathbb{E}[R(a)]$$

fugure1: epsilon = 0.2

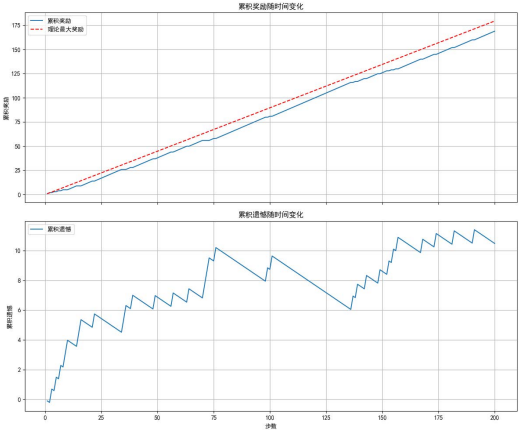
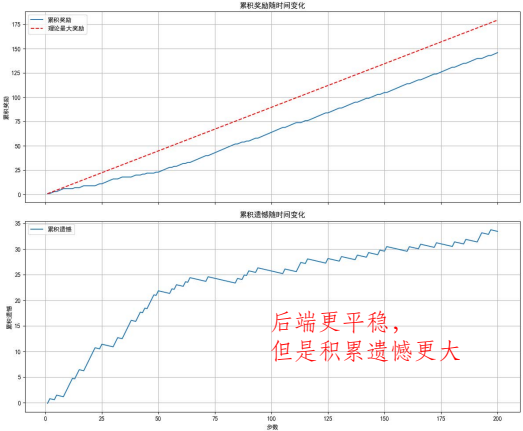


figure2: epsilon = 0.99^steps * 0.5



后端更平稳，
但是积累遗憾更大

UCB algorithm

$$P(\mu > \bar{\mu} + \epsilon \leq exp\{\frac{-n\epsilon^2}{2\sigma^2}\}) \tag{1}$$

To define: $\delta = exp\{\frac{-n\epsilon^2}{2\sigma^2}\}$, $N = 1/\delta$

then we get:

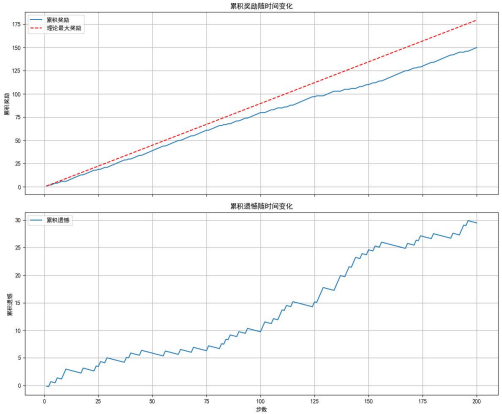
$$P(\mu > \bar{\mu} + \sqrt{\frac{2}{n}lnN} \leq 1/N) \tag{2}$$

- N 是总轮数
- n 是这个杆被选择的次数
- $\bar{\mu} + \sqrt{\frac{2}{n}lnN}$ 是上置信界

在工程上，我们会用一个超参 c 来平衡探索与利用。在第 t 步，选择了动作（杆）a，我们可以得到：

$$UCB_t(a) = \bar{\mu} + c\sqrt{lnN/n} \tag{3}$$

figure3: upper confidence bound



大纲

1. Action Space: Discrete(6)
1. Observation Space: Discrete(500)
2. Import: gymnasium.make("Taxi-v3")

Action Space:

0: Move south (down)
1: Move north (up)
2: Move east (right)
3: Move west (left)
4: Pickup passenger
5: Drop off passenger

Passenger locations:

0: Red
1: Green
2: Yellow
3: Blue
4: In taxi

Destinations:

0: Red
1: Green
2: Yellow
3: Blue

An observation is returned as an int(), calculated by
((taxi_row * 5 + taxi_col) * 5 + passenger_location) * 4 + destination

SARSA Pipeline:

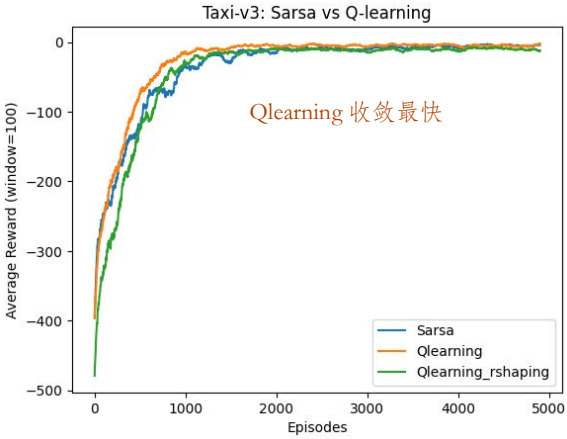
1.创建一个Q table
2.用epsilon-greedy策略走一步
3.再用Q值和epsilon-greedy 找到下一步action
4.用单步TD error更新Qtable
5.online更新

Qlearning Pipeline:

1.创建Q table
2. 用epsilon-greedy策略走一步
3. 再用Q值和epsilon-greedy 找到下一步action
4. 找到next_state 的最大Q值（即使实际action并非对应最大Q值）来更新Q table
5. online 更新

Reward Shaping trick：

1.在Qlearning的基础上增加了
shaping_reward = -dist(taxi, dest)
可以教taxi不要跑的离dest太远



大纲

Make：gymnasium.make(“ALE/Alien-v5”)
Action Space：Discrete(18)
Observation Space：Box(0, 255, (210, 160, 3), uint8)

Wrapper 处理：

1.noop_max=30：reset 时会随机做 0 – 30 个 NOOP，用于打乱起始状态。
2.frame_skip=4：
3.screen_size=84：会把屏幕缩到 84×84。
4.terminal_on_life_loss=False：失去一条命不会把 episode 标记为 terminated（很多实现可选这个行为）。
5.grayscale_obs=True：转灰度。
6.grayscale_newaxis=False：灰度不会被加成单独的最后轴
7.scale_obs=True：把像素归一化到 [0,1]

FrameStackObservation 处理：

1.stack_size=4：堆叠最近 4 帧 → 最终 observation 包含 4 帧历史。
2.padding_type='reset'：在 episode 开始时，空的前帧用 reset 的观测填充（而不是用 0）。
3.vector_entry_point='ale_py.vector_env:AtariVectorEnv' 环境支持 vectorized（并行）版本，用于同时跑多个 env。

堆叠4帧：对应MDP中的（s,a）

DQN network

input : (s,a), in other words, 4-frame obs (N, 4, 84, 84) return : q(s,a)

architecture: 3层卷积 + 一层线性

ReplayBuffer:

1.将经验存储到这里，运用于 offline方法

2.用双端队列deque实现， 满了就剔除最早的经验。

使用DQN有一些特征:

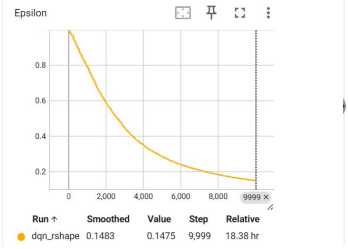
1.loss几乎收敛 2.但是reward依旧很低 3.agent经常会卡在右上角

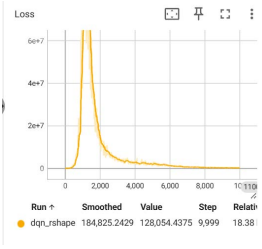
推测原因:

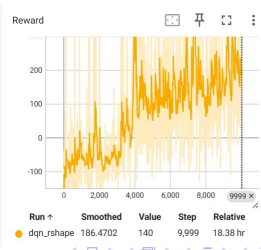
1.loss收敛，但是reward低，应该是收敛到了不好的q table

2.agent卡在右上角：可能是agent采取了保守策略，然后在experience buffer 中不断积累重复经验；并且Q值存在over-estimation bias，通往右上角的Q值被高估， agent会稳定往这边走。

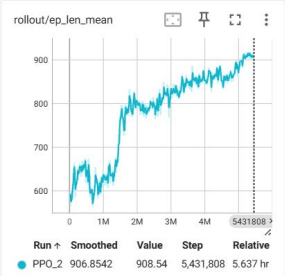
3.dqn的policy的bias比较大（用的 one-step TD 来估计Q值， 并且Qnet和Q*本身也有bias），因此会产生有偏（bias大）且坚定（var小）的动作。

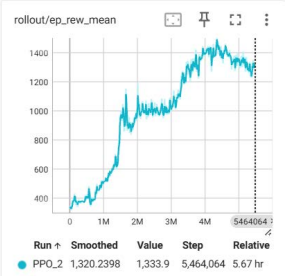


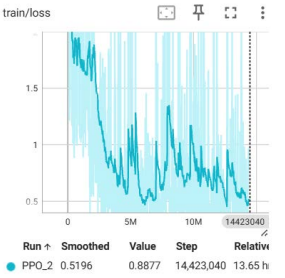




- 反思PPO为什么好：
  - **1.稳定的更新策略：**PPO 通过 clip ratio 约束更新幅度，相当于对 KL 散度做了个软限制，在 Atari 这种高维离散空间里，这种稳定性非常关键。
  - 2.**样本利用率更高:**PPO 在保持 On-policy 性质的前提下，允许在同一批样本上多次梯度更新(importance sampling)。在 Atari 游戏这种 reward 稀疏、环境复杂的场景里，能显著提高学习效率。
  - 3.**优势函数估计（GAE）降低方差:**Atari 游戏奖励很稀疏，直接用 MC 或 TD 估计都会很抖动,PPO 引入 GAE (Generalized Advantage Estimation)，在偏差与方差之间做平衡，使得梯度估计更平滑，更容易学到有效策略。







大纲

论文阅读

Direct Preference Optimization: Your Language Model is Secretly a Reward Model

DPO是现代LLM RLHF RL-post_training的基石，从以往的基于奖励转为优化概率，阅读这一篇文章，回答以下问题：

- 从前面的问题中，我们知道，RL的loss本质上就是最大化奖励，即min（-reward），那么，本论文中的Loss函数是如何由BT（Bradley-Terry）推导来的？优化奖励是如何转化为优化概率的？
- Eq3 中加入了KL散度，解释为什么需要加入KL散度？（如果你有兴趣，可以阅读PPO的论文，详细探索KL散度在RL中的作用）
- 简要说说你认为RL在LLM上可以有什么应用？有人说，RL的本质是去学习Reward Model，基于本文，你觉得RL是否可以学习到Human的偏好Reward？RL是否可以不需要预训练过程完成对LLM的训练，为什么？
- 阅读实验部分，模型经过这种后训练后，对于未经偏好训练的数据输入表现如何？
- （可选）尝试使用DPO解决之前提到的两个问题

Reinforcement Pre-Training

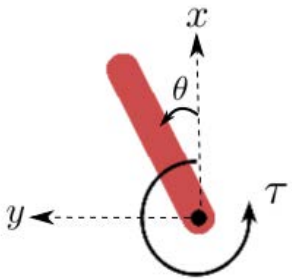
这一篇文章是第一个RL预训练模型， 是RL从后训练角色迈向预训练的一大步，回答以下问题

- 模型的训练数据/语料是什么格式?
- 模型的奖励是如何定义?
- 作者使用了哪些RL方法进行pretrain？了解这些方法并解释其在RLpretrain中的作用和必要性。

大纲

环境介绍

env.obs (3,) #x, y,w
env.act (1,) #T 力矩
action_bound [-2.0, 2.0]



Critic的损失函数：

```
with torch.no_grad():
    target_q = reward + gamma * target_critic(next_state, target_actor(next_state))
q_value = critic(state, action)
critic_loss = F.mse_loss(q_value, target_q)
```

Actor损失函数

```
actor_loss = -critic(state, actor(state)).mean()
```