南開大學
Nankai University

人工智能学院

强化学习实验报告

实验八　基于策略梯度的强化学习方法

姓名：石若川

学号：2111381

专业：智能科学与技术

2024 年 4 月 28 日

1 实验目的

- 学习基于策略梯度的强化学习方法，比较策略梯度方法与行为值函数方法的区别。
- 利用策略梯度方法解决 Gym 库中的 Lunar Lander 环境问题，实现火箭在月球表面的降落。

2 实验原理

2.1 基于策略梯度的强化学习方法

引入记号 $\theta \in \mathbb{R}^{d'}$ 表示策略的参数向量。把在 t 时刻、状态 s 和参数 θ 下选择动作 a 的概率记为 $\pi(a|s,\theta) = \Pr\{A_t = a \mid S_t = s, \theta_t = \theta\}$。如果也使用估计的价值函数，那么价值函数的参数像 $\hat{v}(s,\mathbf{w})$ 中一样用 $\mathbf{w} \in \mathbb{R}^d$ 表示。

策略参数的学习方法都基于某种性能度量 $J(\theta)$ 的梯度，这些梯度是标量 $J(\theta)$ 对策略参数的梯度。这些方法的目标是最大化性能指标，所以它们的更新近似于 J 的梯度上升

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)} \tag{1}$$

其中，$\widehat{\nabla J(\theta_t)} \in \mathbb{R}^{d'}$ 是一个随机估计，它的期望是性能指标对它的参数 θ_t 的梯度的近似。把所有符合这个框架的方法都称为策略梯度法。

2.2 策略梯度方法与行为值函数方法的区别

基于策略梯度的强化学习方法与基于行为值函数的方法有所不同：

- 基于行为值函数的方法：先学习行为值函数，然后根据估计的行为值函数选择动作；如果没有行为值函数的估计，策略也就不会存在。
- 直接学习参数化策略的方法：价值函数仍然可以用于学习策略的参数，但是动作选择不再直接依赖于价值函数。

相较于行为值函数方法，策略梯度的方法具有以下优势：

- 对于离散的动作空间，参数化策略可表示为：

  $$\pi(a|s;\theta) = \frac{e^{h(s,a;\theta)}}{\sum_b e^{h(s,b;\theta)}}$$

  该策略表示可以逼近一个确定性策略，而 $\varepsilon - greedy$ 策略则不能逼近确定性策略。
- 参数化的策略可以逼近任意概率分布，不受行为值函数的限制。
- 策略可以用更简单的函数近似。策略和动作价值函数的复杂度因问题而异。策略复杂度更低的情况下，基于策略的方法一般学习得更快，并且得到更好的渐近策略。
- 策略参数化更容易加入先验知识。

2.3 策略梯度定理

用 τ 表示一组状态-行为序列 $s_0, u_0, \cdots, s_H, u_H$，这一序列的奖励为：

$$R(\tau) = \sum_{t=0}^{H} R(s_t, u_t)$$

目标函数为：

$$U(\theta) = E\left(\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta\right) = \sum_{\tau} P(\tau;\theta) R(\tau)$$

强化学习的目标是找到最优参数 θ 使得：

$$\max_\theta U(\theta) = \max_\theta \sum_{\tau} P(\tau;\theta) R(\tau)$$

目标函数对参数求导：

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta) R(\tau)}{P(\tau;\theta)} \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

利用经验平均估计策略的梯度：

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau;\theta) R(\tau) \tag{2}$$

式2中 $\nabla_\theta \log P(\tau;\theta)$ 称为路径似然率，经过推导可以得到

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)};\theta) &= \nabla_\theta \log \left[\prod_{t=0}^{H} P\left(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}\right) \cdot \pi_\theta\left(u_t^{(i)} | s_t^{(i)}\right) \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log P\left(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}\right) + \sum_{t=0}^{H} \log \pi_\theta\left(u_t^{(i)} | s_t^{(i)}\right) \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log \pi_\theta\left(u_t^{(i)} | s_t^{(i)}\right) \right] \\
&= \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})
\end{aligned} \tag{3}$$

因此，参数的更新可以写为下式的形式，这种更新方法成为 REINFORCE 更新。

$$\theta_{t+1} \doteq \theta_t + \alpha G_t \nabla \log \pi(a_t | s_t; \theta_t) \tag{4}$$

算法的伪代码如下：

REINFORCE：π_* 的模特卡洛策略梯度的控制算法（分幕式）

输入：一个可导的参数化策略 $\pi(a|s,\boldsymbol{\theta})$
算法参数：步长 $\alpha > 0$
初始化策略参数 $\theta \in \mathbb{R}^{d'}$（如初始化为 0）
无限循环 (对于每一幕):
　根据 $\pi(\cdot|\cdot,\theta)$, 生成一幕序列 $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
　对于幕的每一步循环，$t = 0, 1, \ldots, T-1$:
　　$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
　　$\theta \leftarrow \theta + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\theta)$

3　部分代码

定义神经网络结构的代码如下。实验中使用了两个全连接层，两个全连接层之间使用 ReLU 作为激活函数，最后通过 softmax 计算选择动作。

```
self.model = torch.nn.Sequential(
          nn.Linear(input_size, hidden_size, bias=False),
          nn.ReLU(),
          nn.Linear(hidden_size, output_size, bias=False),
          nn.Softmax(dim=-1)
      ).to(device)
```

更新网络的代码如下。首先计算采样轨迹每个状态的奖励，为了降低方差对于每个状态的奖励进行归一化。根据式4的参数更新方法，计算每个动作的对数概率与相应折扣累积回报的乘积，作为损失函数更新网络参数。

```
def backward(self):
      future_reward = 0
      rewards = []
      for r in self.episode_rewards[::-1]:
          future_reward = r + self.gamma * future_reward # 折扣回报
          rewards.append(future_reward)
      rewards = torch.tensor(rewards[::-1], dtype=torch.float32, device=device)
      rewards = (rewards - rewards.mean()) / (rewards.std() +
          np.finfo(np.float32).eps) # 归一化
      loss = torch.sum(torch.mul(self.episode_actions, rewards).mul(-1)) # 损失函数
      self.optimizer.zero_grad()
      loss.backward()
      self.optimizer.step()
      self.reset()
```

训练代码如下。在每次迭代过程中，采样一批轨迹，根据每条轨迹当前的网络和状态选择动作，计

算每条轨迹的奖励并更新网络。每迭代 50 轮保存一次网络模型。

```
while episode<=max_episode:
    observation = env.reset(seed=random_seed, options={})[0]
    done = False
    while not done:
        env.render()
        frame = np.reshape(observation, [1, observation_space])
        action_probs = model.forward(torch.tensor(observation, dtype=torch.float32,
            device=device))
        distribution = Categorical(action_probs) # 计算分布
        action = distribution.sample() # 选择动作
        observation, reward, done, _, _ = env.step(action.item())
        model.episode_actions = torch.cat([model.episode_actions,
            distribution.log_prob(action).reshape(1)])
        model.episode_rewards.append(reward)
        if done:
            batch_rewards.append(np.sum(model.episode_rewards))
            model.backward()
            episode += 1
            if episode % batch_size == 0:
                print('Batch: {}, average reward: {}'.format(episode // batch_size,
                    np.array(batch_rewards).mean()))
                reward_list.append(np.array(batch_rewards).mean())
                batch_rewards = []

            if episode % 50 == 0 and save_directory is not None:
                model.save_checkpoint(save_directory, episode)
```

4　实验结果

实验中所用的 Gym 库为 Lunar Lander 环境。该环境是一个经典的火箭轨迹优化问题，目的是使得火箭能够准确平稳降落在设定好的着陆点（坐标为 $(0,0)$）。该环境基本信息如下：

- 动作空间：Discrete(4)，包括不执行任何操作、启动左方向引擎、启动主引擎、启动右方向引擎。
- 状态空间：表示为一个 8 维向量：着陆器的 x 和 y 坐标、x 和 y 线速度、角度、角速度以及表示每条腿是否与地面接触的两个布尔值。上界为 $[1.51.55.5.3.145.1.1.]$，下界为 $[-1.5-1.5-5.-5.-3.14-5.-0.-0.]$。
- 奖励设置：从屏幕顶部移动到着陆台并停下来的奖励约为 100-140 点。如果着陆器远离着陆点，它就会失去奖励。如果着陆器坠毁，它会获得额外的 -100 分。如果它停下来，它会额外获得 +100 分。每条与地面接触的腿 +10 分。主机点火每帧-0.3 分。每帧启动侧引擎为 -0.03 分。成功解决问题获得 200 分。

实验中设定批量大小 $batchsize = 10$，学习率 $lr = 10^{-4}$，折扣系数 $\gamma = 0.99$，$\varepsilon = 10^{-4}$，最大迭代次数 2500，优化器为 Adam 优化器。经过 2500 次迭代后，每个批次的平均奖励的变化曲线如图4.1所示。在训练初期，智能体随机选取动作奖励较低。随着训练次数增加，奖励逐渐上升。在迭代 1500 次后，奖励收敛到 200 左右。

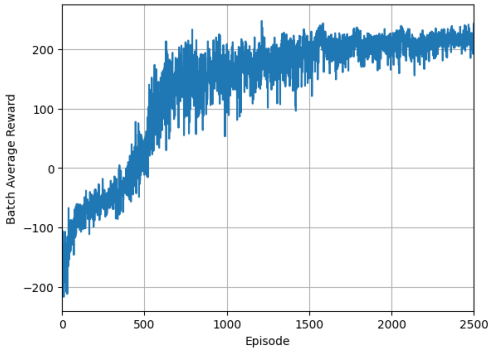

图 4.1: 批平均奖励随训练的变化曲线

利用训练 2500 次得到的模型 checkpoint_25000.pth 进行测试，测试结果如表1所示，图4.2展示了一次的测试结果，测试视频见附件 test.mp4。测试结果说明，根据该模型着陆器能够准确平稳地在着陆点附近着陆。

表 1: 测试结果

测试次数	奖励值
1	261.64
2	258.41
3	257.35
4	259.15
5	293.97
6	249.64
7	280.22
8	251.97
9	226.37
10	243.47
平均值	258.22

图 4.2: 一次测试的结果

5 实验总结

通过这次实验，我学习了基于梯度策略的强化学习方法，并利用 REINFORCE 方法解决了 Gym 库中的 Lunar Lander 问题。

基于策略梯度的强化学习方法是一类与值函数方法不同的强化学习方法，其核心思想是直接学习策略函数（即如何选择动作的函数），而不是学习值函数。这种方法适用于离散或连续动作空间的问题，并且能够有效地应对高度不确定性和复杂性的环境。

实验中在 Lunar Lander 环境上对 REINFORCE 更新方法进行了测试。测试结果表明，经过 2500 次的迭代，REINFORCE 方法可以成功解决 Lunar Lander 问题。