机器学习大作业

git实现多人编程

使用工具：git、VScode

【给傻子的Git教程】https://www.bilibili.com/video/BV1Hkr7YYEh8?vd_source=23274d00140aafc65734bc29f0c6864b

【和傻子一起写代码】https://www.bilibili.com/video/BV1udEuzrEa7?vd_source=23274d00140aafc65734bc29f0c6864b

如何使用 Git 进行多人协作开发（全流程图解）_git多人协作开发流程-CSDN博客

模拟风力扰动

加在 `ax` 上一个扰动项：`ax += wind_force / mass`

模拟燃料消耗

- 每次喷气时减少燃料；
- 质量逐渐减小，影响加速度；
- 如剩余燃料越多，奖励越高。

添加风力参数和初始燃料

```
self.wind_enabled = True
self.wind_force_max = 3.0  # 单位 N，最大横向风力

self.mass_init = 100.0      # 火箭总质量（可调整）
self.fuel_mass = 90.0       # 可燃烧燃料
self.fuel_consumption_rate = 0.02  # 每次推力所耗 kg
```

加入风力扰动和质量影响

```
# 计算当前质量
mass = self.mass_init - self.fuel_mass
mass = max(mass, 10.0)  # 防止质量为负

# 风力扰动
wind_force = 0.0
if self.wind_enabled:
    wind_force = np.random.uniform(-self.wind_force_max, self.wind_force_max)
self._last_wind_force = wind_force  # 保存当前风速，用于绘图

ax = (fx + wind_force - rho*vx) / mass
ay = (fy - self.g - rho*vy) / mass
```

加入燃料消耗

$$\Delta m = \dot{m} = \alpha \cdot f/g$$

$$m_{fuel}(t + \Delta t) = max(0, m_{fuel}(t) - \dot{m})$$

f：当前推力（单位 N）

g：重力加速度（约 9.8 m/s²）

α：燃料消耗速率因子（单位 kg/“重力单位推力”）

$\dot{m}$：当前时间步的燃料消耗量

m_{fuel}：剩余燃料质量

推力越大，燃烧速度越快；

推力以“g”为单位标准化（使其与火箭本身抗重力能力相关）；

```
# 推力消耗燃料
if f > 0:
    self.fuel_mass -= self.fuel_consumption_rate * (f / self.g)  # 简单按推力归一化计算
    self.fuel_mass = max(self.fuel_mass, 0)
```

将剩余燃料加入 reward

```
if self.task == 'landing' and self.already_landing:
    reward += 0.1 * (self.fuel_mass / 30.0)
```

燃料耗尽判失败

```
if self.fuel_mass <= 0 and not self.already_landing:
    self.already_crash = True
```

状态向量扩展：加入 fuel_ratio 与 step_ratio

`flatten()` 函数：

```
x = np.array([...]) / 100.
fuel_ratio = np.array([self.fuel_mass / self.mass_init], dtype=np.float32)
step_ratio = np.array([state['t'] / self.max_steps], dtype=np.float32)
return np.concatenate([x, fuel_ratio, step_ratio])
```

在 `__init__()` 结尾设置：

```
self.state_dims = 10  # 原为8，现在加入两个额外维度
```

图像界面实时显示风速与燃料

`draw_text()` 函数中末尾加入：

```
pt = (10, 120)
text = "fuel left: %.2f kg" % self.fuel_mass
put_text(canvas, text, pt)

pt = (10, 140)
if self.wind_enabled:
    text = "wind force: %.2f N" % self._last_wind_force
else:
    text = "wind force: OFF"
put_text(canvas, text, pt)
```

Rocket 初始化方式更新

```
env = Rocket(task='hover', max_steps=800, rocket_type='falcon',
             wind_enabled=True,
             wind_force_max=2.5,
             fuel_mass=120.0,
             mass_init=140.0,
             fuel_consumption_rate=0.02)
```

让转动惯量随质量变化

$$I = \frac{1}{12} \cdot m(t) \cdot H^2$$

I：火箭绕中心轴的转动惯量（单位 kg·m²）

$m(t)$：当前火箭总质量，随燃料减少而减小

H：火箭高度

```
mass = max(self.mass_init - self.fuel_mass, 10.0)
I = (1/12) * mass * (self.H ** 2)
atheta = ft * self.H/2 / I
```

非对称风力作用（风引起转动）

当前模型默认火箭为质量均匀的竖直矩形刚体，质心在几何中心（重心）处，即火箭中点、高度 $H/2$ 位置。

设定风的施力点相对于质心的偏移为：

$$h_{\text{wind}} \sim \mathcal{U}(-H/2, H/2)$$

我们希望风力不仅推动火箭平移，也能吹歪火箭，引发转动（角加速度）

$$\tau_{wind} = F_{wind} \cdot h_{\text{wind}}$$

则

$$\alpha_{\theta,wind} = \frac{\tau_{wind}}{I}$$

```
        mass = max(self.mass_init - self.fuel_mass, 10.0)
        I = (1/12) * mass * (self.H ** 2)  # 更新转动惯量

        tau_engine = ft * self.H/2 # 计算推力产生的角加速度
        # 引入风力随机扰动点位
        self.h_wind = np.random.uniform(-self.H/2, self.H/2)
        tau_wind = wind_force * self.h_wind
        atheta = (tau_engine + tau_wind) / I
```

在 `draw_text()` 中增加：

```
pt = (10, 180)
put_text(canvas, "wind @ h = %.1f m" % h_wind, pt)
```

训练之后第一次reward比之前好了很多，我不是很懂，但是gpt这么说

🧮 一、修改后模型的“推力使用效率变高了”

🔍 你引入了如下机制：

1. **推力会消耗燃料**；
2. **燃料消耗后质量减小** → 同样推力产生更高加速度；
3. **转动惯量减小** → 更容易控制姿态；
4. **着陆 reward 被放大（残余燃料 × 剩余步数）**；
5. **风力扰动产生“小扰动自稳定”效果**（意外的训练帮助）。

✅ 三、总结对比

特性	原始模型	修改后模型
推力转换	有方向计算	保留
加速度计算	没有除以质量	✅ 除以动态质量
质量变化	❌ 固定隐式常量	✅ 动态更新 (初始质量 - 燃料质量)
燃料消耗	❌ 无	✅ 每次推力减少燃料
控制挑战性	🚨 极高，易崩溃	✅ 稳定渐进
学习稳定性	❌ 非常差，reward 初期极低	✅ 初期易收敛

❗ 问题：没有质量 = 推力再大也不会变“笨”

在真实物理中：

$$a = \frac{F_{\text{net}}}{m}$$

如果质量 m 很大，加速度应很小。

但原模型中没这个除法，推力直接决定加速度：

> 💥 所以推力再大，火箭立刻获得高速 → 非常容易失控 → 训练时频繁坠毁。

解决问题：每轮开始时火箭“油量是上轮剩下的”

```
def __init__(...):
    ...
    self.fuel_mass_init = fuel_mass  # <--- 记录初始燃料
    self.fuel_mass = fuel_mass
    ...
```

reset()添加

```
self.fuel_mass = self.fuel_mass_init
```

这样就不会燃料突然消失然后非常吓人了

```
  File "<frozen importlib._bootstrap_external>", line 1130, in get_data
KeyboardInterrupt
(base) PS F:\Tsinghua\major\senior_2\machine_learning\term_project\rocket-recycling-main> conda activate rocket-env
(rocket-env) PS F:\Tsinghua\major\senior_2\machine_learning\term_project\rocket-recycling-main> python example_train.py
episode id: 0, episode reward: 107.051
episode id: 1, episode reward: 108.354
episode id: 2, episode reward: 109.554
episode id: 3, episode reward: 121.471
episode id: 4, episode reward: 57.366
episode id: 5, episode reward: 170.571
episode id: 6, episode reward: 161.427
episode id: 7, episode reward: 149.950
episode id: 8, episode reward: 69.327
episode id: 9, episode reward: 110.186
episode id: 10, episode reward: 169.310
episode id: 11, episode reward: 60.348
episode id: 12, episode reward: 50.482
episode id: 13, episode reward: 42.066
episode id: 14, episode reward: 155.882
episode id: 15, episode reward: 44.811
episode id: 16, episode reward: 105.651
episode id: 17, episode reward: 117.127
episode id: 18, episode reward: 54.345
episode id: 19, episode reward: 42.979
episode id: 20, episode reward: 77.854
episode id: 21, episode reward: 134.329
episode id: 22, episode reward: 84.239
episode id: 23, episode reward: 67.377
episode id: 24, episode reward: 85.743
```

policy.py

原来的policy代码保存在副本里了

Entropy Loss

鼓励策略在训练初期保持对动作的多样性探索。强化学习常常面临“早收敛”的问题，策略在尚未充分尝试所有可能动作之前就锁定在某个次优策略上，导致泛化能力差。通过对策略输出的动作分布计算熵值，并在损失函数中给予一定权重的正向奖励，可以有效防止策略过早变得过于保守，使其在面对复杂环境扰动（如风力、燃料变化）时仍具备探索能力，从而学到更稳健的控制策略。

```
entropy = -(log_probs * torch.exp(log_probs)).sum()
actor_loss = (-log_probs * advantage.detach()).mean() - 0.001 * entropy
```

Layer Normalization（层归一化）

提升训练过程的稳定性

```
    def __init__(self, input_dim, output_dim):
        super().__init__()
```

```python
        self.mapping = PositionalMapping(input_dim=input_dim, L=7)

        h_dim = 128
        # tyq
        self.linear1 = nn.Linear(self.mapping.output_dim, h_dim)
        self.norm1 = nn.LayerNorm(h_dim)
        self.linear2 = nn.Linear(h_dim, h_dim)
        self.norm2 = nn.LayerNorm(h_dim)
        self.linear3 = nn.Linear(h_dim, h_dim)
        self.norm3 = nn.LayerNorm(h_dim)
        self.linear4 = nn.Linear(h_dim, output_dim)
        self.relu = nn.LeakyReLU(0.2)

        # self.linear1 = nn.Linear(in_features=self.mapping.output_dim, out_features=h_dim,
bias=True)
        # self.linear2 = nn.Linear(in_features=h_dim, out_features=h_dim, bias=True)
        # self.linear3 = nn.Linear(in_features=h_dim, out_features=h_dim, bias=True)
        # self.linear4 = nn.Linear(in_features=h_dim, out_features=output_dim, bias=True)
        # self.relu = nn.LeakyReLU(0.2)

    def forward(self, x):
        # shape x: 1 x m_token x m_state
        # x = x.view([1, -1])
        # x = self.mapping(x)
        # x = self.relu(self.linear1(x))
        # x = self.relu(self.linear2(x))
        # x = self.relu(self.linear3(x))
        # x = self.linear4(x)
        # tyq
        x = x.view([1, -1])
        x = self.mapping(x)
        x = self.relu(self.norm1(self.linear1(x)))
        x = self.relu(self.norm2(self.linear2(x)))
        x = self.relu(self.norm3(self.linear3(x)))
        x = self.linear4(x)
        return x
```

“推力变化惯性”机制

模拟现实中火箭发动机推力不是瞬时切换的，而是有惯性，改变推力时会渐进调整。

$$f_{t+1} = \beta \cdot f_{t0} + (1-\beta) \cdot f_{target}, \beta \in [0.8, 0.98]$$

f_{target}：策略当前选择的推力

f_t：当前真实推力值

$\beta \in [0.8, 0.98]$：惯性权重

```python
        # tyq 推力惯性
        # f, vphi = self.action_table[action]
        f_target, vphi = self.action_table[action]

        # 推力惯性参数
        self._throttle_beta = 0.9 if not hasattr(self, '_throttle_beta') else self._throttle_beta
        self.f = self.f if hasattr(self, 'f') else f_target  # 初始化上次推力

        # 平滑更新推力（模拟推力惯性）
        self.f = self._throttle_beta * self.f + (1 - self._throttle_beta) * f_target
        f = self.f
```

可视化 reward 组成与训练过程

calculate_reward

```python
        # 保存各个reward分量
        landing_bonus = 0.0
        crash_penalty = 0.0
        fuel_bonus = 0.0

        v = (state['vx'] ** 2 + state['vy'] ** 2) ** 0.5
        if self.task == 'landing' and self.already_crash:
            reward = (reward + 5*np.exp(-1*v/10.)) * (self.max_steps - self.step_id)
            reward = crash_penalty
        if self.task == 'landing' and self.already_landing:
            reward = (1.0 + 5*np.exp(-1*v/10.))*(self.max_steps - self.step_id)
            reward = landing_bonus
            fuel_bonus = 0.1 * (self.fuel_mass / 30.0)
            reward += fuel_bonus

        self._last_reward_parts = {
        'dist_reward': float(dist_reward),
        'pose_reward': float(pose_reward),
        'fuel_bonus': float(fuel_bonus),
        'landing_bonus': float(landing_bonus),
        'crash_penalty': float(crash_penalty),
        'total_reward': float(reward),
        'fuel_left': float(self.fuel_mass),
        'step_id': self.step_id,
        'landed': self.already_landing,
        'crashed': self.already_crash
        }
```

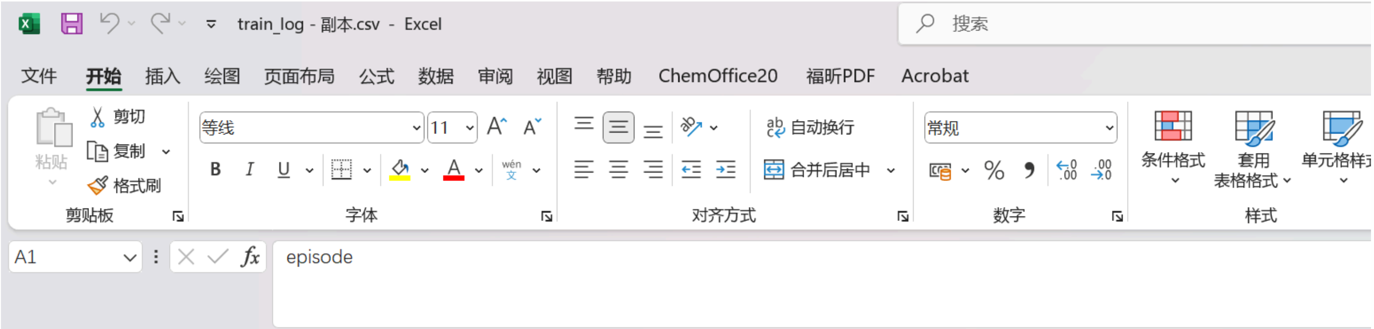
example_train.py

```python
 which device we want to run on
 torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

e__ == '__main__':

 = 'hover'  # 'hover' or 'landing'
et_type = 'falcon'  #SunYunru:考虑变量rocket_type:可以选'falcon'或'starship'
ion = '_raw'  #SunYunru:增设变量version，方便对比不同修改下代码运行结果
rd_video = True  #SunYunru:增设变量record_video，确定是否保存视频

m_episode = 20000  #SunYunru:改到20000轮训练
steps = 800

 = Rocket(task=task, max_steps=max_steps, rocket_type=rocket_type)
q
= Rocket(task=task, max_steps=max_steps, rocket_type=rocket_type,
     wind_enabled=True, wind_force_max=2.0,
     mass_init=120.0, fuel_mass=100.0)

_folder = os.path.join('./', task + '_' + rocket_type + version + '_ckpt')
ot os.path.exists(ckpt_folder):
os.mkdir(ckpt_folder)

# 记录奖励情况 tyq
path = os.path.join(ckpt_folder, 'train_log.csv')
ot os.path.exists(log_path):
with open(log_path, 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['episode', 'reward', 'dist', 'pose', 'fuel_bonus', 'landing_bonus', 'crash_penalty', 'fuel_left', 'step', 'landed', 'crashed'])

_episode_id = 0
RDS = []

= ActorCritic(input_dim=env.state_dims, output_dim=env.action_dims).to(device)
```

```python
REWARDS.append(np.sum(rewards))
print('episode id: %d, episode reward: %.3f'
      % (episode_id, np.sum(rewards)))

# tyq 记录奖励
reward_parts = env._last_reward_parts if hasattr(env, '_last_reward_parts') else {}

with open(log_path, 'a', newline='') as f:
    writer = csv.writer(f)
    writer.writerow([
        episode_id,
        reward_parts.get('total_reward', 0),
        reward_parts.get('dist_reward', 0),
        reward_parts.get('pose_reward', 0),
        reward_parts.get('fuel_bonus', 0),
        reward_parts.get('landing_bonus', 0),
        reward_parts.get('crash_penalty', 0),
        reward_parts.get('fuel_left', 0),
        reward_parts.get('step_id', 0),
        reward_parts.get('landed', False),
        reward_parts.get('crashed', False)
    ])

if episode_id % 100 == 1:
    plt.figure()
```

csv数据含义：

列名	含义
episode	第几轮训练
reward	当前 episode 的总 reward（最终得分）
dist	与目标点的距离惩罚（分数越小越好）
pose	姿态角惩罚（倾斜越大惩罚越大）
fuel_bonus	着陆后保留燃料获得的奖励
landing_bonus	成功着陆获得的大额奖励
crash_penalty	墜毁情况下的惩罚性奖励（乘以剩余步数）
fuel_left	剩余燃料量（kg）
step	本 episode 实际执行的步数（最大为 800）
landed	是否成功着陆
crashed	是否坠毁



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	episode	reward	dist	pose	fuel_bonus	landing_bc	crash_pena	fuel_left	step	landed	crashed			
2	0	0	0.058861	-0.34362	0	0	0	84.45703	689	FALSE	TRUE			
3	1	0	0.061834	-0.34109	0	0	0	82.67283	746	FALSE	TRUE			
4	2	0	0.054243	-0.29876	0	0	0	85.37599	639	FALSE	TRUE			
5	3	0	0.0678	-0.42139	0	0	0	82.3322	739	FALSE	TRUE			
6	4	0	0.050594	-0.28501	0	0	0	84.45716	631	FALSE	TRUE			
7	5	0	0.060916	-0.39408	0	0	0	84.02602	680	FALSE	TRUE			
8	6	0	0.065317	-0.415	0	0	0	82.30677	753	FALSE	TRUE			
9	7	0	0.057625	-0.39131	0	0	0	83.16378	671	FALSE	TRUE			
10	8	0	0.069176	-0.45256	0	0	0	83.74235	702	FALSE	TRUE			
11	9	0	0.052647	-0.30109	0	0	0	84.96465	666	FALSE	TRUE			
12	10	0	0.066048	-0.39946	0	0	0	83.77553	698	FALSE	TRUE			
13	11	0	0.067197	-0.38085	0	0	0	85.31686	669	FALSE	TRUE			
14	12	0	0.054435	-0.28447	0	0	0	84.52883	706	FALSE	TRUE			
15	13	0	0.055754	-0.42369	0	0	0	85.60506	636	FALSE	TRUE			
16	14	0	0.053387	-0.3525	0	0	0	86.15082	621	FALSE	TRUE			
17	15	0	0.063488	-0.37174	0	0	0	86.17906	663	FALSE	TRUE			
18	16	0	0.059768	-0.35385	0	0	0	86.23806	652	FALSE	TRUE			
19	17	0	0.053137	-0.34528	0	0	0	86.33537	600	FALSE	TRUE			
20	18	0	0.065843	-0.37247	0	0	0	84.38082	704	FALSE	TRUE			
21	19	0	0.065462	-0.32973	0	0	0	84.8332	705	FALSE	TRUE			
22	20	0	0.065973	-0.43951	0	0	0	84.92471	674	FALSE	TRUE			
23	21	0	0.0648	-0.40852	0	0	0	85.8542	673	FALSE	TRUE			
24	22	0	0.05515	-0.35477	0	0	0	86.79211	608	FALSE	TRUE			
25	23	0	0.061376	-0.41157	0	0	0	85.01543	667	FALSE	TRUE			
26	24	0	0.055905	-0.37921	0	0	0	87.33989	583	FALSE	TRUE			
27	25	0	0.060193	-0.40115	0	0	0	84.78758	667	FALSE	TRUE			