REINFORCEMENT LEARNING

阿里云 | TIANCHI天池

阿里云天池龙珠计划

ALIBABA CLOUD TIANCHI DRAGON BALL PROJECT

免费CPU/GPU算力　　完整学练赛体系

天池TOP选手答疑　　天池认证奖励

强化学习训练营

带你从0利用PPO算法玩超级玛丽

天池龙珠计划是阿里云天池平台针对人工智能初学者提供的完整入门学习体系，其目标是为学习者奠定人工智能知识体系的学习基础，通过理论学习+项目实践的方式构建学习者对人工智能的整体认知。

学习路径

体系化学习内容

学习体系提供**3大**基础训练营，**3大**进阶训练营，**5大**应用学习赛，**上百道**测试题以及竞赛资源。

DSW算力

随时随地在线编程

为每一位学习者提供在线编程实践环境，可支持随时随地**代码实践**，充分降低学习门槛。

Kol带学

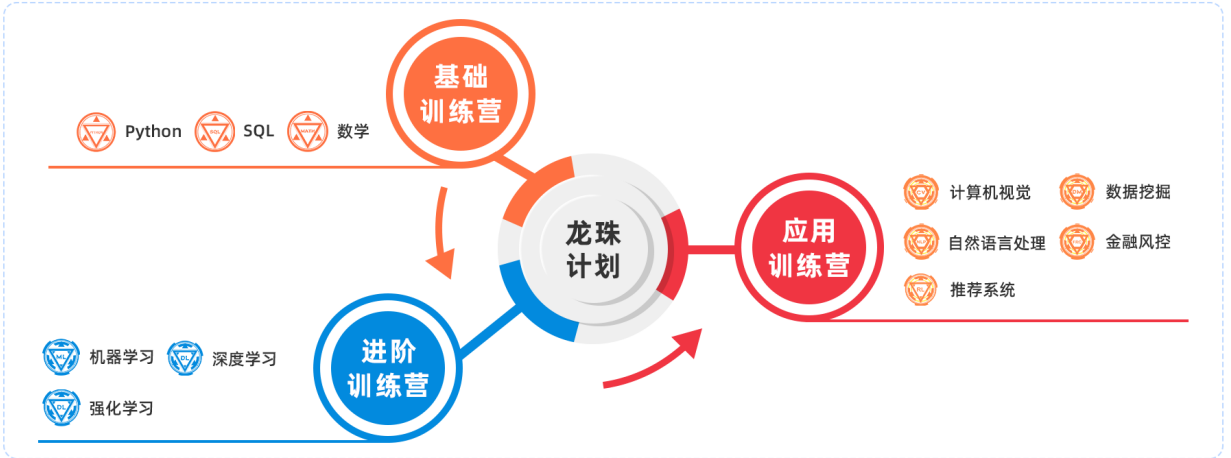
天池Kol在线答疑

天池Top选手分享学习经验，为学习者答疑解惑。社区构建完整的用户体系，面向所有学习者开放。

学习奖励

阿里云官方结业证书

每个训练营提供**训练营挑战证书**，通关六大训练营得龙珠计划系列勋章和纸质证书。



本学习体系由天池与开源学习组织Datawhale共同维护

天 池 龙 珠 计 划

强化训练营

阿里云天池龙珠计划之强化学习训练营是为学习者准备的强化学习零基础入门训练营，包含以下4个任务：

Task01 强化学习：实战PPO通关超级玛丽。

Task02 强化学习：PPO算法环境配置

Task03 强化学习：PPO算法实战练习

Task04 强化学习：学习赛实战

经过整个学习，学习者可以掌握强化学习基础，并且对于强化学习能做什么有一个基本的概念，本训练营从实战入手，学习者学习过程更有动力，能让学习者更快的走向人工智能之路。

主办方

TIANCHI天池

大 数 据 众 智 平 台

扫码回复：强化训练营，

即可进行学习获取证书

0.1 前言介绍
0.0.1 一、安装Python包
0.0.2 二、加载训练好的模型
0.0.3 2 强化学习初体验
0.0.3.1 value-base
0.0.3.2 policy-base
0.0.3.3 ppo
0.0.3.4 ppo的工程实现
0.0.4 3 加入强化学习训练营
0.0.5 4 超神进阶
0.0.5.1 增加辅助信息
0.0.5.2 算法升级
0.1 为什么是ppo?
0.1 强化学习是什么?
0.1 R0 ppo玩超级玛丽1-1关的视频
0.1 R1 先来学习如何用代码实现随机动作play超级玛丽游戏（5 min）
0.0.1 R1.1 通过网页播放出来刚才的运行实况
0.0.2 R1.2 随机动作play超级玛丽的完整代码
0.0.2.1 番外篇，用键盘玩超级玛丽：
0.1 R2 完整代码通关play超级玛丽(10 min)
0.0.0.2 本地运行方法：本地可以使用docker一键运行，docker的好处是已经包含了环境，可直接运行。
0.1 R3 认识环境
0.0.1 R3.1 （由浅入深）倒立摆环境（carplot）讲解
0.0.2 R3.2 超级玛丽环境讲解
0.0.3 R3.3常用env Wrapper技巧
0.0.3.1 R3.3.1 rgb图像转灰度图
0.0.3.2 R3.3.2 SkipFrame
0.0.3.3 R3.3.2 CustomReward
0.1 R4 PPO（近段策略优化）算法讲解
0.0.1 R4.1 由浅入深，简化版ppo（100行代码）
0.0.2 R4.2 openai版本ppo算法实践 （训练超级玛丽）
0.0.3 R4.2.1 设计DL模型
0.0.4 R4.2.2 设计ppo的ac模型（Actor/Critic）
0.0.5 R4.2.3 定义ppo 算法
0.0.5.0.1 R4.2.3.1 定义PPOBuffer 用来存储交互数据，提供给模型训练使用
0.0.5.1 R4.2.3.2 定义ppo算法 及 更新策略
0.0.5.2 R4.2.3.3 PPO算法完整代码（添加log记录、mpi多进程）
0.0.5.3 R4.2.3.3 主函数

0.0.5.4 R4.2.3.4 查看训练过程指标
0.0.5.5 R4.2.3.5 加载训练好的模型并在游戏中运行
0.1 R5 强化学习的近况&挑战
0.0.1 R5.1强化学习的近况
0.0.2 R5.2强化学习的挑战

0.1 前言介绍

玛丽奥作为一代人的童年，陪伴了我们的成长。

如今随着深度学习、强化学习等技术的发展，越来越多的游戏正在被AI征服，那么今天我们一起来从0开始，试着征服超级玛丽吧！

0.0.1 一、安装Python包

安装超级玛丽软件包，`gym_super_mario_bros`这个包已经帮我们把游戏的rom封装好了常用的python接口，环境中输入下面pip命令即可安装。

```
pip install gym gym_super_mario_bros


# 利用阿里云镜像安装
pip install -i https://mirrors.aliyun.com/pypi/simple/ gym gym_super_mario_bros
```

用python代码控制马里奥执行随机动作

```
import gym_super_mario_bros
from gym import wrappers
env = wrappers.Monitor(env,"./gym-results", force=True)
#执行5000个简单的向右随机操作
done = True #游戏结束标志
for step in range(5000):
    if done:
        #如果游戏结束则重置:
        state = env.reset()
    state, reward, done, info = env.step(env.action_space.sample())
# 关闭创建的游戏env
env.close()
```

然后查看`./gym-results`文件夹下的mp4文件,即为上面代码执行的过程录像。

tips: 也可以在下面网页上直接尝试运行上面代码并可视化运行效果。

```
https://colab.research.google.com/drive/1ULu0t__0LPl5oI528oNvoOhs50XTZ2Nf?usp=sharing
```

0.0.2 二、加载训练好的模型

可视化ppo算法在游戏中的表现

```
get_action = load_pytorch_policy(args.fpath)
o, r, d, ep_ret, ep_len, n = env.reset(), 0, False, 0, 0, 0
hidden = (torch.zeros((1, 512), dtype=torch.float).to(device), torch.zeros((1, 512),
dtype=torch.float).to(device))
while n < num_episodes:
    if render:
        env.render()
        time.sleep(1e-3)
    a, hidden = get_action(o, hidden)
    o, r, d, _ = env.step(a.numpy().item())
    ep_ret += r
    ep_len += 1

    if d or (ep_len == max_ep_len):
        print('Episode %d \t EpRet %.3f \t EpLen %d' % (n, ep_ret, ep_len))
        o, r, d, ep_ret, ep_len = env.reset(), 0, False, 0, 0
        hidden=(torch.zeros((1, 512), dtype=torch.float).to(device), torch.zeros((1, 512),
dtype=torch.float).to(device))
        n += 1
```

tips: 使用docker 直接运行：

```
docker run --gpus all -v /tmp/.X11-unix:/tmp/.X11-unix registry.cn-
shanghai.aliyuncs.com/tcc-public/super-mario-ppo:localdisplay
```

如果不会使用docker的话，也可以下载项目源码，自己动手运行。

```
GitHub：https://github.com/gaoxiaos/Supermariobros-PPO-pytorch
```

0.0.3 2 强化学习初体验

强化学习的思想在生活中很常见，核心思想就是通过不断的尝试来让自己学会某件事情，因此强化学习的方法大致分为两类（`value-base`和`Policy-base`）。

0.0.3.1 value-base

顾名思义，`value-base`就是**“以价值为导向”**的优化方法，比如游戏中在某个状态下执行不同的action得到不同的价值**“value”**，我们向着value高的方向去优化，那么最终就可以得到一个**“每次都大概率选择最高价值的动作执行”**的模型。

0.0.3.2 policy-base

同样的字面意思，`policy-base`则是**“以策略为导向”**的优化方法，agent每次选择action时都以当前策略模型比优化前**“好多少”**为优化方向，**“好多少”**也有可能是负数（惩罚优化），好的越多那么优化权重（力度）就越大。在很多连续动作空间问题上policy-base方法经常使用，比如本文用到的ppo（近端策略优化）算法就是policy-base方法。

0.0.3.3 ppo

`ppo`作为强化学习领域研究的benchmark，往往代表了当前强化学习在待研究场景下达到的基准最好状态。然后在此基础上研究能够超过ppo基准的优化方法。所以熟练掌握ppo是我们当前想要在强化学习做算法研究的必备技能之一。

ppo全名**Proximal Policy Optimization**，翻译过来**“近端策略优化”**，策略优化的思想就是上面policy-base方法，而“近端”，也就是“限制新旧策略更新幅度，保证新旧策略要在相近状态”，这样可以保障模型策略更新稳定收敛，否则可能会出现严重抖动难以收敛等问题。

0.0.3.4 ppo的工程实现

目前网上常见的主要有baselines版、spiningup、莫烦版本、rllib、天授等，每个在实现方式上都不太一样，导致初学者往往会对工程实现上有很多迷惑，比如有的版本Actor和Critic是分离的而有的是共用的，loss计算有的是分开单独做梯度优化的也有的是加在一起做梯度优化，还有“近端约束”有的计算kl散度约束有的直接clip到固定范围，但其实这些不同的实现方式也都在常规的gym环境下达到了很好的效果，所以很长一段时间并没有论文证明ac共享权重好还是分开优化更好，直到今年procgen的发布（procgen旨在解决以往强化学习对环境的记忆>泛化能力而开发的动态环境benchmark），openai对比了不同的工程实现方式，并提出了ppg算法方法，其实就是ac分开优化的同时，增加辅助阶段，用于ac之间共享权重，所以整体来说ac分离与否各有优缺点。所以本文的代码实现为了顺应ppg的发展，将ac分开设计独立优化，构建简化版本方便初学者理解，这样到学习ppg时则仅需在此基础上添加辅助阶段即可。

0.0.4 3 加入强化学习训练营

钉钉扫下方二维码进入训练营学习交流群和小伙伴们交流学习

今天只是借助超级玛丽做简单的入门初体验，如果你觉得这样的形式对你有用，可以查看完整版强化学习训练营（https://tianchi.aliyun.com/specials/promotion/aicamprl），也欢迎入群（添加wx:pythonbrief）和其他同样的小伙伴交流学习，后续会有更多项目实践文章或视频。

0.0.5 4 超神进阶

如果你并不满足现状，想做一些超神操作通关所有关卡（目前ppo通关超级玛丽最高记录是29/32,剩余未通关的是4-4、7-4和8-4，因为这几关不仅考验类似走迷宫的能力还要有严谨的进退策略，所以单靠ppo是非常难的），这里给你提供一些参考路径。

0.0.5.1 增加辅助信息

在某些特别难的关卡现有gym_super_mario_bros抽取出来的游戏环境（状态、action等）不能满足你的需求，这个时候你可以自己添加，其实类似gym_super_mario_bros这样的包或者openai的retro把游戏转化为代码可操作的rl-env都是通过py虚拟机加载游戏RAM，然后针对这个游戏解析RAM的按键（寄存器）地址、状态地址等，比如retro已经解析支持常见的千种游戏解析（但是需要自己添加游戏ram，这里分享一个童年游戏ram集散地：http://www.atarimania.com/）

那么我们想要获得更多的信息，就可以通过查询寄存器地址，自己添加字段来获取信息或执行特殊技能，如超级玛丽部分寄存器地址：

Table

RAM	Information
0x0000	Temp/various uses. Is used in vertical physics for gravity acceleration (value copied from 0x0709).
0x0001	Player's animation
0x0002	Temp/various. Something to do with player y (but it skips to 0 every x frames, even when you dont move)
0x0003	Player's direction (and others). • 1 – Right • 2 – Left
0x0004	How much to load
0x0005	Something to do with player x (same as 0x0002)
0x0008	Object Offset.
0x0009	Frame Counter. Count number of Frames.
0x000A	Button state AB (flags) • 0x00 – No Button • 0x40 – A • 0x80 – B • 0xC0 – Both
0x000B	Vertical direction input state (flags); • 0x00 – No Button • 0x40 – Down • 0x80 – Up • 0xC0 – Both
	Player's state • 0x00 – Leftmost of screen • 0x01 – Climbing vine

0.0.5.2 算法升级

经过上面的探索，你会发现ppo的不同实现方式会有不同的表现，且各有优缺点，比如ac加在一起做loss优化时两者的权重很难定义有可能训练过程中会相互干扰，带来负面的影响，而不共享权重则在复杂的环境下很难训练导致难以收敛，所以在尝试不同的工程效果之后，可以升级ppo算法为ppg（Phasic Policy Gradient），兼顾独立优化的好处外增加一个阶段用于ac间共享权重。官方源码（https://github.com/openai/phasic-policy-gradient）对初学者可读性较差，添加（wx:pythonbrief）后续为大家分享ppg简化版本代码。

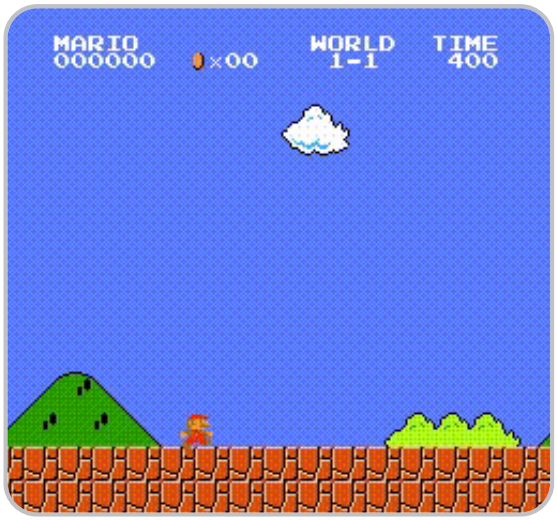
0.1 为什么是ppo?

ppo算法作为强化学习领域out of art的算法，如果你要学习强化学习的话 ppo会是你最常用的算法。openai早已把ppo 作为自己的默认算法，所以我希望你能认真学完ppo算法并为自己所用。

0.1 强化学习是什么?

简单来说 强化学习是一类通过不断与环境交互来学习如何达到设定目标的一类算法，比如走迷宫，传统的运筹学算法往往是通过遍历所有的点来完成路径规划，而强化学习则是实现一个anget,让这个
agent自己去随机探索路线，在探索的过程中学习如何走的更远并最终走到终点，这就是强化学习的思想。

0.1 R0 ppo玩超级玛丽1-1关的视频



0.1 R1 先来学习如何用代码实现随机动作play超级玛丽游戏（5 min）

本次学习需要的相关库如下。

1. gym
2. gym_super_mario_bros
3. opencv-python
4. spinup
5. joblib

注意本次学习需要在`GPU`环境中，所以进入DSW后，点击右侧的环境切换按钮，运行模式选择`GPU`，如下图所示。

如果有任何问题可以扫下方二维码添加钉钉学习交流群进行学习交流：

```
# 安装相关包
# 再次提醒：必须先切换环境为GPU
!pip install gym gym_super_mario_bros opencv-python spinup joblib --user
```

```
#导入实验需要的包
from nes_py.wrappers import JoypadSpace
import gym_super_mario_bros
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT
```

```
#使用gym_super_mario_bros包函数创建游戏环境env
env = gym_super_mario_bros.make('SuperMarioBros-v0')
```

```
#指定环境为简单模式（动作简化，去除一些左上、左下等复杂动作）
env = JoypadSpace(env, SIMPLE_MOVEMENT)
```

```
#使用gym的wrapper函数对游戏视频进行录像（由于notebook不支持display，我们录像后播放观看）

from gym import wrappers
env = wrappers.Monitor(env,"./gym-results", force=True)
```

```
#执行5000个简单的向右随机操作
done = True #游戏结束标志
for step in range(5000):
    if done:
        #如果游戏结束则重置:
        state = env.reset()
    state, reward, done, info = env.step(env.action_space.sample())
```

```
# 关闭创建的游戏env
env.close()
```

注：此处主要对如何在代码中运行游戏有个感知，每个env相关的函数参数意义详见R3.2章节环境讲解部分

0.0.1 R1.1 通过网页播放出来刚才的运行实况

```
import io
import base64
from IPython.display import HTML

video = io.open('./gym-results/openaigym.video.%s.video000000.mp4' % env.file_infix,
'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''
    <video width="360" height="auto" alt="test" controls><source src="data:video/mp4;base64,
{0}" type="video/mp4" /></video>'''
.format(encoded.decode('ascii')))
```

0.0.2 R1.2 随机动作play超级玛丽的完整代码

```
from nes_py.wrappers import JoypadSpace
import gym_super_mario_bros
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT

#借助包gym_super_mario_bros创建
env = gym_super_mario_bros.make('SuperMarioBros-v0')
env = JoypadSpace(env, SIMPLE_MOVEMENT)
from gym import wrappers
env = wrappers.Monitor(env,"./gym-results", force=True)

done = True
for step in range(5000):
    if done:
        state = env.reset()
    state, reward, done, info = env.step(env.action_space.sample())
#     env.render()

env.close()
```

```
#通过网页播放出来刚才的运行实况
import io
import base64
from IPython.display import HTML

video = io.open('./gym-results/openaigym.video.%s.video000000.mp4' % env.file_infix,
'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''
    <video width="360" height="auto" alt="test" controls><source src="data:video/mp4;base64,
{0}" type="video/mp4" /></video>'''
.format(encoded.decode('ascii')))
```

0.0.2.1 番外篇，用键盘玩超级玛丽：

gym_super_mario_bros -e -m < `human` or `random` >

0.1 R2 完整代码通关play超级玛丽(10 min)

```
#下载我提前训练好的权重和代码
!git clone https://github.com/gaoxiaos/Supermariobros-PPO-pytorch.git
!cd auper_ppo
```

```
#运行play测试程序
!python test.py
```

```
#查看运行录像
video = io.open('./gym-results/openaigym.video.%s.video000000.mp4' % env.file_infix,
'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''
    <video width="360" height="auto" alt="test" controls><source src="data:video/mp4;base64,
{0}" type="video/mp4" /></video>'''
.format(encoded.decode('ascii')))
```

0.0.0.2 本地运行方法：本地可以使用docker一键运行，docker的好处是已经包含了环境，可直接运行。

link：supermrio.readthedoc.com

0.1 R3 认识环境

强化学习中的环境就等同于深度学习或者数据挖掘课题的“数据”，强化学习通过与环境交互来产生数据，所以对环境的认知直接关系到最终结果的好坏，在很多强化学习的研究和竞赛里往往对环境的trick比算法的改进效果更为明显。

0.0.1 R3.1 （由浅入深）倒立摆环境（carplot）讲解

运行一个倒立摆环境（CartPole） 观察环境返回什么 环境的动作有哪些?

#补一个倒立摆的GIF图

```
#创建倒立摆'CartPole-v0' env
import gym
env = gym.make('CartPole-v0')
```

```
#初始化游戏环境
env.reset()
```

从上面的返回可以看到我们在执行环境初始化\重置时env返回给我们了初始化后的环境状态为：

[0.03749292, -0.03226631, 0.01609263, -0.04661368]

这四个数字组成的状态变量（state variables）分别含义如下：

0.03749292： 小车在轨道上的位置（position of the cart on the track）

-0.03226631： 杆子与竖直方向的夹角（angle of the pole with the vertical）

0.01609263： 小车速度（cart velocity）

-0.04661368： 角度变化率（rate of change of the angle）

```
#环境包含的动作有哪些?
print("env.action_space: ", env.action_space)
```

从结果来看动作空间为2，也就是说倒立摆这个环境只有两个动作可以操作，分别是0和1 （向左和向右）从倒立摆的动画不难理解，通过左右移动来保持倒立摆不倒。

```
#执行一个向左的操作
obj, reward, done, info = env.step(1) #1 向右 0向左
print("obj", obj)
print("reward", reward)
print("done", done)
print("info", info)
```

一个动作执行后，环境会返回四个变量（obj:新的状态（对照前面环境初始化的状态理解）、reward：指定该动作获得的奖励值（在游戏中的得分）、done:回合是否结束（你控制的小人是不是死了，对应回合结束）、info:额外信息（该游戏较简单，info为空））

```
#随机获取一个动作
action = env.action_space.sample()
print(action)
```

通过sample（）函数可以快速得到一个随机动作，由于该游戏动作空间为2，所以sample得到的值为0或1

```
# Virtual display
from pyvirtualdisplay import Display

virtual_display = Display(visible=0, size=(1400, 900))
virtual_display.start()
```

```
#运行1000组随机动作
env = gym.make('CartPole-v0')
from gym import wrappers
env = wrappers.Monitor(env,"./", force=True)
env.reset()
for _ in range(1000):
    env.render() #服务器上无display,不支持render
    obj, reward, done, info = env.step(env.action_space.sample()) # take a random action
    if done:
        env.reset()
env.close()



# from IPython import display
# import matplotlib
# import matplotlib.pyplot as plt
# %matplotlib inline

# env = gym.make('CartPole-v0')
# env.reset()
# img = plt.imshow(env.render(mode='rgb_array')) # only call this once
# for _ in range(100):
#     img.set_data(env.render(mode='rgb_array')) # just update the data
#     display.display(plt.gcf())
#     display.clear_output(wait=True)
#     action = env.action_space.sample()
#     env.step(action)
```

```
import gym
env = gym.make('CartPole-v0')
env = wrappers.Monitor(env,"./gym-results")#, force=True
env.reset()
for _ in range(1000):
#     env.render() #服务器上无display,不支持render
    obj, reward, done, info = env.step(env.action_space.sample()) # take a random action
    if done:
        env.reset()
env.close()
```

0.0.2 R3.2 超级玛丽环境讲解

超级玛丽主要区别于倒立摆游戏的是超级玛丽的obj观测值（状态）为当前帧图片（像素），和人类玩超级玛丽一致，通过观察每一帧图像（大脑/模型）输出要执行的action

```
#创建env
from nes_py.wrappers import JoypadSpace
import gym_super_mario_bros
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT

#借助包gym_super_mario_bros创建
env = gym_super_mario_bros.make('SuperMarioBros-v0')
```

注：SuperMarioBros---v

其中：

是{1，2，3，4，5，6，7，8}中的一个数字，表示世界

是{1，2，3，4}中的一个数字，表示一个世界中的阶段

是{0，1，2，3}中的一个数字，指定要使用的rom模式

0：标准ROM

1:降采样ROM

2：像素rom

3：矩形ROM

```
#初始化env
obj = env.reset()
print(obj.shape)
```

由输出可以看到超级玛丽的观测值变成了一张240*256的rgb图片

为了验证，我们可视化出来

```
import matplotlib.pyplot as plt
plt.imshow(obj)
```

```
#接下来看一下动作空间
print("env.action_space: ", env.action_space)
```

默认情况下， gym_super_mario_bros环境使用完整的NES操作空间256 离散动作。为了解决这个问题，gym_super_mario_bros.actions提供 三个操作列表（RIGHT_ONLY、SIMPLE_MOVEMENT和COMPLEX_MOVEMENT） 对于nes_py.wrappers.JoypadSpace包装器

```
#我们选用SIMPLE_MOVEMENT来看下是否满足我们的通关需求
env = JoypadSpace(env, SIMPLE_MOVEMENT)
print("env.action_space: ", env.action_space)
```

7个基本动作包含了常用的操作 如上下左右，跳跃，右+跳，左+跳。由此其实已经基本满足了常用的操作，而选择更多的动作反而会增加模型学习的难度。所以我们选择SIMPLE_MOVEMENT模式即可

```
#随机执行一个操作
obj, reward, done, info = env.step(1) #这里随机选择执行动作1
print("obj.shape", obj.shape)
print("reward", reward)
print("done", done)
print("info", info)
```

强化学习执行step动作的返回一般是标准的，所以这里的返回同前面的倒立摆，动作执行后，环境返回四个变量（obj:新的观测值（一帧rgb图片）、reward：执行该动作获得的奖励值（在游戏中的得分）、done:回合是否结束（你控制的小人是不是死了，对应回合结束）、info:额外信息（比如'life': 2，剩余2条命等））

详细字段解释，参见https://www.cnpython.com/pypi/gym-super-mario-bros

```
##留一个空位 看下是否讲解reward
```

0.0.3 R3.3常用env Wrapper技巧

```
#先重新引入下相关包，防止报错
import gym_super_mario_bros
from gym.spaces import Box
from gym import Wrapper
from nes_py.wrappers import JoypadSpace#BinarySpaceToDiscreteSpaceEnv
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT, COMPLEX_MOVEMENT, RIGHT_ONLY
import cv2
import numpy as np
import subprocess as sp
```

0.0.3.1 R3.3.1 rgb图像转灰度图

想象一下你在玩超级玛丽时如果把彩色图像换成灰度图，其实对你的操作并没有多大影响（只要能看出来障碍物即可判断路线和动作），反而在模型训练中，rgb图像对算力和训练时间的要求会成倍增长，所以综合考虑咱们转换成灰度图才输入网络

```
#借助cv2即（opencv）包快速转换COLOR_RGB2GRAY
def process_frame(frame):
    if frame is not None:
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY) #图像转换
        frame = cv2.resize(frame, (84, 84))[None, :, :] / 255. #裁剪合适大小，并归一化
        return frame
    else:
        return np.zeros((1, 84, 84))
```

0.0.3.2 R3.3.2 SkipFrame

由于超级玛丽等游戏开发是面向玩家的（人），而非电脑，所以面向人类通关设计时，很多游戏帧是被放慢了，比如执行一个action并不会立刻得到reard而是在接下来的几帧里才逐渐成效，换个通俗的说法，其实这么快速的游戏帧对我们并不需要，我们只需要每秒能看到几帧就足以通关了，所以我们根据经验，每四帧只取一帧即可

```
class CustomSkipFrame(Wrapper):
    def __init__(self, env, skip=4):
        super(CustomSkipFrame, self).__init__(env)
```

```
        self.observation_space = Box(low=0, high=255, shape=(4, 84, 84))
        self.skip = skip

    def step(self, action):
        total_reward = 0
        states = []
        state, reward, done, info = self.env.step(action)
        for i in range(self.skip):
            if not done:
                state, reward, done, info = self.env.step(action)
                total_reward += reward
                states.append(state)
            else:
                states.append(state)
        states = np.concatenate(states, 0)[None, :, :, :]
        return states.astype(np.float32), reward, done, info

    def reset(self):
        state = self.env.reset()
        states = np.concatenate([state for _ in range(self.skip)], 0)[None, :, :, :]
        return states.astype(np.float32)
```

0.0.3.3 R3.3.2 CustomReward

强化学习的优化目标必须是可量化的，所以在游戏里我们直接的优化目标就是最大化reward,但是很多时候游戏直接设定的reward并不完全切合我们的实际目的（比如通关），或者在某个特定场景下（关卡下）不合适，所以越是复杂的游戏场景，越是需要自定义reward来进行修正。

这里我们做了几个小优化如下：

1.reward += (info["score"] - self.curr_score) / 40.

原来的reward仅包含了对“离终点更近”的奖励和“时间消耗”、”死掉“的惩罚

为了让游戏更好玩，我们添加了info["score"]，包含了对获得技能、金币的奖励，但不是重点，为了不影响整体要通关的属性，弱化他

2.if done:

```
            if info["flag_get"]:
                reward += 50
            else:
                reward -= 50
```

我们对回合结束时到达终点和未达到的奖励和惩罚进行放大，激励agent更快速的到达终点

3.这里仅仅是对reward修改的一些示例，后面自己在实战时可以自己根据实际情况进行定义，比如当agent有时陷入一个错误的路线卡住时，可以添加一个缓冲区让agent学会后退等

```
class CustomReward(Wrapper):
    def __init__(self, env=None):
        super(CustomReward, self).__init__(env)
        self.observation_space = Box(low=0, high=255, shape=(1, 84, 84))
        self.curr_score = 0

    def step(self, action):
        state, reward, done, info = self.env.step(action)
        state = process_frame(state)
        reward += (info["score"] - self.curr_score) / 40.
        self.curr_score = info["score"]
        if done:
            if info["flag_get"]:
                reward += 50
            else:
                reward -= 50
        return state, reward / 10., done, info

    def reset(self):
        self.curr_score = 0
        return process_frame(self.env.reset())
```

```
#至此，我们完成了超级玛丽环境的自定义，封装如下：
def create_train_env(world, stage, action_type, output_path=None):
    env = gym_super_mario_bros.make("SuperMarioBros-{}-{}-v0".format(world, stage))
    if action_type == "right":
        actions = RIGHT_ONLY
    elif action_type == "simple":
        actions = SIMPLE_MOVEMENT
    else:
        actions = COMPLEX_MOVEMENT
    env = JoypadSpace(env, actions)
    env = CustomReward(env)
    env = CustomSkipFrame(env)
    return env, env.observation_space.shape[0], len(actions)
```

```
#测试一下
custom_env = create_train_env(1,1,'simple')
print(custom_env)
```

0.1 R4 PPO（近段策略优化）算法讲解

1、策略（要输出最优动作的策略模型）

2、近端（代理函数的剪裁）

3、优化（使用代理函数）的出现及其实际意义，导致了算法的命名。

0.0.1 R4.1 由浅入深，简化版ppo（100行代码）

```
#导入gym和torch相关包
import gym
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical
```

```
#Hyperparameters
learning_rate = 0.0005 #学习率
gamma         = 0.98   #
lmbda         = 0.95
eps_clip      = 0.1
K_epoch       = 3
T_horizon     = 20
```

```
#定义PPO架构
class PPO(nn.Module):
    def __init__(self):
        super(PPO, self).__init__()
        self.data = [] #用来存储交互数据

        self.fc1   = nn.Linear(4,256) #由于倒立摆环境简单，这里仅用一个线性变换来训练数据
        self.fc_pi = nn.Linear(256,2) #policy函数（输出action）的全连接层
        self.fc_v  = nn.Linear(256,1) #value函数（输出v）的全连接层
        self.optimizer = optim.Adam(self.parameters(), lr=learning_rate) #优化器

    #policy函数
    #输入观测值x
    #输出动作空间概率，从而选择最优action
    def pi(self, x, softmax_dim = 0):
        x = F.relu(self.fc1(x))
        x = self.fc_pi(x)
        prob = F.softmax(x, dim=softmax_dim)
        return prob

    #value函数
    #输入观测值x
    #输出x状态下value的预测值（reward），提供给policy函数作为参考值
```

```
    def v(self, x):
        x = F.relu(self.fc1(x))
        v = self.fc_v(x)
        return v

    #把交互数据存入buffer
    def put_data(self, transition):
        self.data.append(transition)

    #把数据形成batch，训练模型时需要一个一个batch输入模型
    def make_batch(self):
        s_lst, a_lst, r_lst, s_prime_lst, prob_a_lst, done_lst = [], [], [], [], [], []
        for transition in self.data:
            s, a, r, s_prime, prob_a, done = transition

            s_lst.append(s)
            a_lst.append([a])
            r_lst.append([r])
            s_prime_lst.append(s_prime)
            prob_a_lst.append([prob_a])
            done_mask = 0 if done else 1
            done_lst.append([done_mask])

        s,a,r,s_prime,done_mask, prob_a = torch.tensor(s_lst, dtype=torch.float),
torch.tensor(a_lst), \
                                          torch.tensor(r_lst), torch.tensor(s_prime_lst,
dtype=torch.float), \
                                          torch.tensor(done_lst, dtype=torch.float),
torch.tensor(prob_a_lst)
        self.data = []
        return s, a, r, s_prime, done_mask, prob_a


    #训练模型

    def train_net(self):
        #make batch 数据，喂给模型
        s, a, r, s_prime, done_mask, prob_a = self.make_batch()
```

```
        for i in range(K_epoch): #K_epoch：训练多少个epoch
            #计算td_error 误差，value模型的优化目标就是尽量减少td_error
            td_target = r + gamma * self.v(s_prime) * done_mask
            delta = td_target - self.v(s)
            delta = delta.detach().numpy()

            #计算advantage:
            #即当前策略比一般策略（baseline）要好多少
            #policy的优化目标就是让当前策略比baseline尽量好，但是每次更新时又不能偏离太多，所以后面会有个
clip
            advantage_lst = []
            advantage = 0.0
            for delta_t in delta[::-1]:
                advantage = gamma * lmbda * advantage + delta_t[0]
                advantage_lst.append([advantage])
            advantage_lst.reverse()
            advantage = torch.tensor(advantage_lst, dtype=torch.float)

            #计算ratio 防止单词更新偏离太多
            pi = self.pi(s, softmax_dim=1)
            pi_a = pi.gather(1,a)
            ratio = torch.exp(torch.log(pi_a) - torch.log(prob_a))  # a/b == exp(log(a)-
log(b))

            #通过clip 保证ratio在（1-eps_clip, 1+eps_clip）范围内
            surr1 = ratio * advantage
            surr2 = torch.clamp(ratio, 1-eps_clip, 1+eps_clip) * advantage
            #这里简化ppo，把policy loss和value loss放在一起计算
            loss = -torch.min(surr1, surr2) + F.smooth_l1_loss(self.v(s) ,
td_target.detach())

            #梯度优化
            self.optimizer.zero_grad()
            loss.mean().backward()
            self.optimizer.step()
```

```
#主函数：简化ppo 这里先交互T_horizon个回合然后停下来学习训练，再交互，这样循环10000次
def main():
    #创建倒立摆环境
```

```
    env = gym.make('CartPole-v1')
    model = PPO()
    score = 0.0
    print_interval = 20

    #主循环
    for n_epi in range(10000):
        s = env.reset()
        done = False
        while not done:
            for t in range(T_horizon):
                #由当前policy模型输出最优action
                prob = model.pi(torch.from_numpy(s).float())
                m = Categorical(prob)
                a = m.sample().item()
                #用最优action进行交互
                s_prime, r, done, info = env.step(a)

                #存储交互数据，等待训练
                model.put_data((s, a, r/100.0, s_prime, prob[a].item(), done))
                s = s_prime

                score += r
                if done:
                    break

            #模型训练
            model.train_net()

        #打印每轮的学习成绩
        if n_epi%print_interval==0 and n_epi!=0:
            print("# of episode :{}, avg score : {:.1f}".format(n_epi,
score/print_interval))
            score = 0.0

    env.close()

if __name__ == '__main__':
    main()
```

0.0.2 R4.2 openai版本ppo算法实践 （训练超级玛丽）

通过上面简化版本的ppo玩倒立摆，你已经对ppo有了简单的认知，但是上面的做法还有很多待改进的地方，比如模型太简单，如果像超级玛丽这种观测值是图像的话，简单的线性变换肯定不满足条件

、策略模型和value模型应该分开优化损失，因为policy和value的loss很多情况下数值是差别很大的，小的那个往往得不到有效优化等

接下来介绍openai 官方版本的ppo怎么实现

```
#3.然后我们来设计ppo算法来实现马里奥通关
#3.1 先创建游戏环境（
#    a.组合定义action
#    b.重定义reward
#    c.堆叠zhenlv
#    d.预处理输入的图像
#    ）
```

```
#导入相关包
import gym_super_mario_bros
from gym.spaces import Box
from gym import Wrapper
from nes_py.wrappers import JoypadSpace#BinarySpaceToDiscreteSpaceEnv
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT, COMPLEX_MOVEMENT, RIGHT_ONLY
import cv2
import numpy as np
import subprocess as sp
```

```
class Monitor:
    def __init__(self, width, height, saved_path):

        self.command = ["ffmpeg", "-y", "-f", "rawvideo", "-vcodec", "rawvideo", "-s", "
{}X{}".format(width, height),
                        "-pix_fmt", "rgb24", "-r", "80", "-i", "-", "-an", "-vcodec",
"mpeg4", saved_path]
        try:
            self.pipe = sp.Popen(self.command, stdin=sp.PIPE, stderr=sp.PIPE)
```

```
        except FileNotFoundError:
            pass

    def record(self, image_array):
        self.pipe.stdin.write(image_array.tostring())


def process_frame(frame):
    if frame is not None:
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        frame = cv2.resize(frame, (84, 84))[None, :, :] / 255.
        return frame
    else:
        return np.zeros((1, 84, 84))


class CustomReward(Wrapper):
    def __init__(self, env=None, monitor=None):
        super(CustomReward, self).__init__(env)
        self.observation_space = Box(low=0, high=255, shape=(1, 84, 84))
        self.curr_score = 0
        if monitor:
            self.monitor = monitor
        else:
            self.monitor = None

    def step(self, action):
        state, reward, done, info = self.env.step(action)
        if self.monitor:
            self.monitor.record(state)
        state = process_frame(state)
        reward += (info["score"] - self.curr_score) / 40.
        self.curr_score = info["score"]
        if done:
            if info["flag_get"]:
                reward += 50
            else:
                reward -= 50
        return state, reward / 10., done, info
```

```
    def reset(self):
        self.curr_score = 0
        return process_frame(self.env.reset())


class CustomSkipFrame(Wrapper):
    def __init__(self, env, skip=4):
        super(CustomSkipFrame, self).__init__(env)
        self.observation_space = Box(low=0, high=255, shape=(4, 84, 84))
        self.skip = skip

    def step(self, action):
        total_reward = 0
        states = []
        state, reward, done, info = self.env.step(action)
        for i in range(self.skip):
            if not done:
                state, reward, done, info = self.env.step(action)
                total_reward += reward
                states.append(state)
            else:
                states.append(state)
        states = np.concatenate(states, 0)[None, :, :, :]
        return states.astype(np.float32), reward, done, info

    def reset(self):
        state = self.env.reset()
        states = np.concatenate([state for _ in range(self.skip)], 0)[None, :, :, :]
        return states.astype(np.float32)


def create_train_env(world, stage, action_type, output_path=None):
    env = gym_super_mario_bros.make("SuperMarioBros-{}-{}-v0".format(world, stage))
    if output_path:
        monitor = Monitor(256, 240, output_path)
    else:
        monitor = None
    if action_type == "right":
```

```
        actions = RIGHT_ONLY
    elif action_type == "simple":
        actions = SIMPLE_MOVEMENT
    else:
        actions = COMPLEX_MOVEMENT
    env = JoypadSpace(env, actions)
    env = CustomReward(env, monitor)
    env = CustomSkipFrame(env)
    return env, env.observation_space.shape[0], len(actions)
```

```
#5.创建ppo算法
#5.1创建ac
```

```
import numpy as np
import scipy.signal
from gym.spaces import Box, Discrete

import torch
import torch.nn as nn
from torch.distributions.normal import Normal
from torch.distributions.categorical import Categorical
import torch.nn.functional as F
```

```
def combined_shape(length, shape=None):
    if shape is None:
        return (length,)
    return (length, shape) if np.isscalar(shape) else (length, *shape)
```

```
#定义通用cnn model类
class cnn_model(nn.Module):
    def __init__(self, num_inputs, num_out, activation=nn.ReLU):
        super(cnn_model, self).__init__()
        self.conv1 = nn.Conv2d(num_inputs, 32, 3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.conv4 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.lstm = nn.Linear(32 * 6 * 6, 512)
        # self.critic_linear = nn.Linear(512, 1)
        # self.actor_linear = nn.Linear(512, num_actions)
        self.fc_out = nn.Linear(512, num_out)
        self._initialize_weights()
```

```
    def _initialize_weights(self):
        for module in self.modules():
            if isinstance(module, nn.Conv2d) or isinstance(module, nn.Linear):
                nn.init.xavier_uniform_(module.weight)
                # nn.init.kaiming_uniform_(module.weight)
                nn.init.constant_(module.bias, 0)
            elif isinstance(module, nn.LSTMCell):
                nn.init.constant_(module.bias_ih, 0)
                nn.init.constant_(module.bias_hh, 0)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = x.view(x.size(0), -1)
        x = F.relu(self.lstm(x))
        out = self.fc_out(x)
        return out
```

```
#utils:
def count_vars(module):
    return sum([np.prod(p.shape) for p in module.parameters()])

def discount_cumsum(x, discount):
    return scipy.signal.lfilter([1], [1, float(-discount)], x[::-1], axis=0)[::-1]
```

```
class userActor(nn.Module):

    def __init__(self, obs_dim, act_dim, hidden_sizes, activation):
        super().__init__()
        self.logits_net = cnn_model(obs_dim, act_dim, activation=activation)
        print(self.logits_net)

    def forward(self, obs, act=None):
        pi = Categorical(logits=self.logits_net(obs))
        logp_a = None
        if act is not None:
            logp_a = pi.log_prob(act)
        return pi, logp_a
```

```
class userCritic(nn.Module):

    def __init__(self, obs_dim, hidden_sizes, activation):
        super().__init__()
        self.v_net = cnn_model(obs_dim, 1, activation=activation)#cnn_net([obs_dim] +
list(hidden_sizes) + [1], activation)
        print(self.v_net)

    def forward(self, obs):
        return torch.squeeze(self.v_net(obs), -1) # Critical to ensure v has right shape.
```

```
#5.3定义ppo
```

```
import numpy as np
import torch
import torch.nn as nn
from torch.optim import Adam
import gym
import time
import scipy.signal
# import spinup.algos.pytorch.ppo.core as core
# from core_1 import Actor, Critic
from core import userCritic, userActor
from env import create_train_env
from spinup.utils.logx import EpochLogger
from spinup.utils.mpi_pytorch import setup_pytorch_for_mpi, sync_params, mpi_avg_grads
from spinup.utils.mpi_tools import mpi_fork, mpi_avg, proc_id, mpi_statistics_scalar,
num_procs

device = torch.device('cuda')
```

```
class PPOBuffer:
    """
    A buffer for storing trajectories experienced by a PPO agent interacting
    with the environment, and using Generalized Advantage Estimation (GAE-Lambda)
    for calculating the advantages of state-action pairs.
    """

    def __init__(self, obs_dim, act_dim, size, gamma=0.99, lam=0.95):
        self.obs_buf = np.zeros(combined_shape(size, obs_dim), dtype=np.float32)
```

```
        self.act_buf = np.zeros(combined_shape(size, act_dim), dtype=np.float32)
        self.adv_buf = np.zeros(size, dtype=np.float32)
        self.rew_buf = np.zeros(size, dtype=np.float32)
        self.ret_buf = np.zeros(size, dtype=np.float32)
        self.val_buf = np.zeros(size, dtype=np.float32)
        self.logp_buf = np.zeros(size, dtype=np.float32)
        self.gamma, self.lam = gamma, lam
        self.ptr, self.path_start_idx, self.max_size = 0, 0, size

    def store(self, obs, act, rew, val, logp):
        """
        Append one timestep of agent-environment interaction to the buffer.
        """
        assert self.ptr < self.max_size     # buffer has to have room so you can store
        self.obs_buf[self.ptr] = obs
        self.act_buf[self.ptr] = act
        self.rew_buf[self.ptr] = rew
        self.val_buf[self.ptr] = val
        self.logp_buf[self.ptr] = logp
        self.ptr += 1

    def finish_path(self, last_val=0):
        """
        Call this at the end of a trajectory, or when one gets cut off
        by an epoch ending. This looks back in the buffer to where the
        trajectory started, and uses rewards and value estimates from
        the whole trajectory to compute advantage estimates with GAE-Lambda,
        as well as compute the rewards-to-go for each state, to use as
        the targets for the value function.

        The "last_val" argument should be 0 if the trajectory ended
        because the agent reached a terminal state (died), and otherwise
        should be V(s_T), the value function estimated for the last state.
        This allows us to bootstrap the reward-to-go calculation to account
        for timesteps beyond the arbitrary episode horizon (or epoch cutoff).
        """

        path_slice = slice(self.path_start_idx, self.ptr)
        rews = np.append(self.rew_buf[path_slice], last_val)
```

```
        vals = np.append(self.val_buf[path_slice], last_val)

        # the next two lines implement GAE-Lambda advantage calculation
        deltas = rews[:-1] + self.gamma * vals[1:] - vals[:-1]
        self.adv_buf[path_slice] = discount_cumsum(deltas, self.gamma * self.lam)

        # the next line computes rewards-to-go, to be targets for the value function
        self.ret_buf[path_slice] = discount_cumsum(rews, self.gamma)[:-1]

        self.path_start_idx = self.ptr

    def get(self):
        """
        Call this at the end of an epoch to get all of the data from
        the buffer, with advantages appropriately normalized (shifted to have
        mean zero and std one). Also, resets some pointers in the buffer.
        """
        assert self.ptr == self.max_size    # buffer has to be full before you can get
        self.ptr, self.path_start_idx = 0, 0
        # the next two lines implement the advantage normalization trick
        adv_mean, adv_std = mpi_statistics_scalar(self.adv_buf)
        self.adv_buf = (self.adv_buf - adv_mean) / adv_std
        data = dict(obs=self.obs_buf, act=self.act_buf, ret=self.ret_buf,
                    adv=self.adv_buf, logp=self.logp_buf)
        #data.to(device)
        return {k: torch.as_tensor(v, dtype=torch.float32).to(device) for k,v in
data.items()}

def ppo(env_fn, actor=nn.Module, critic=nn.Module, ac_kwargs=dict(), seed=0,
        steps_per_epoch=4000, epochs=50, gamma=0.99, clip_ratio=0.2, pi_lr=3e-4,
        vf_lr=1e-3, train_pi_iters=80, train_v_iters=80, lam=0.97, max_ep_len=1000,
        target_kl=0.01, logger_kwargs=dict(), save_freq=10):
    # Special function to avoid certain slowdowns from PyTorch + MPI combo.
    setup_pytorch_for_mpi()

    # Set up logger and save configuration
    logger = EpochLogger(**logger_kwargs)
    logger.save_config(locals())
```

```
    # Random seed
    seed += 10000 * proc_id()
    torch.manual_seed(seed)
    np.random.seed(seed)

    # Instantiate environment
    env = env_fn()
    obs_dim = env.observation_space.shape
    act_dim = env.action_space.n

    # Create actor-critic module
    ac_pi = actor(obs_dim[0], act_dim, hidden_sizes=[64, 64], activation=nn.Tanh)  #
env.observation_space, env.action_space, nn.ReLU)
    ac_v = critic(obs_dim[0], hidden_sizes=[64, 64], activation=nn.Tanh)  #
env.observation_space, nn.ReLU)

    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cup')
    ac_pi.to(device)
    ac_v.to(device)

    # Sync params across processes
    sync_params(ac_pi)
    sync_params(ac_v)

    # Count variables
    def count_vars(module):
        return sum([np.prod(p.shape) for p in module.parameters()])
    var_counts = tuple(count_vars(module) for module in [ac_pi, ac_v])
    logger.log('\nNumber of parameters: \t pi: %d, \t v: %d\n'%var_counts)

    # Set up experience buffer
    local_steps_per_epoch = int(steps_per_epoch / num_procs())
    buf = PPOBuffer(obs_dim, env.action_space.shape, local_steps_per_epoch, gamma, lam)

    # Set up function for computing PPO policy loss
    def compute_loss_pi(data):
        obs, act, adv, logp_old = data['obs'], data['act'], data['adv'], data['logp']

        # Policy loss
```

```
    pi, logp = ac_pi(obs, act)
    ratio = torch.exp(logp - logp_old)
    clip_adv = torch.clamp(ratio, 1-clip_ratio, 1+clip_ratio) * adv
    loss_pi = -(torch.min(ratio * adv, clip_adv)).mean()

    # Useful extra info
    approx_kl = (logp_old - logp).mean().item()
    ent = pi.entropy().mean().item()
    clipped = ratio.gt(1+clip_ratio) | ratio.lt(1-clip_ratio)
    clipfrac = torch.as_tensor(clipped, dtype=torch.float32).mean().item()
    pi_info = dict(kl=approx_kl, ent=ent, cf=clipfrac)

    return loss_pi, pi_info

# Set up function for computing value loss
def compute_loss_v(data):
    obs, ret = data['obs'], data['ret']
    return ((ac_v(obs) - ret)**2).mean()

# Set up optimizers for policy and value function
pi_optimizer = Adam(ac_pi.parameters(), lr=pi_lr)
vf_optimizer = Adam(ac_v.parameters(), lr=vf_lr)

# Set up model saving
logger.setup_pytorch_saver(ac_pi)

def update():
    data = buf.get()

    pi_l_old, pi_info_old = compute_loss_pi(data)
    pi_l_old = pi_l_old.item()
    v_l_old = compute_loss_v(data).item()

    # Train policy with multiple steps of gradient descent
    for i in range(train_pi_iters):
        pi_optimizer.zero_grad()
        loss_pi, pi_info = compute_loss_pi(data)
        kl = mpi_avg(pi_info['kl'])
        if kl > 1.5 * target_kl:
```

```
            logger.log('Early stopping at step %d due to reaching max kl.'%i)
            break
        loss_pi.backward()
        mpi_avg_grads(ac_pi)    # average grads across MPI processes
        pi_optimizer.step()

    logger.store(StopIter=i)

    # Value function learning
    for i in range(train_v_iters):
        vf_optimizer.zero_grad()
        loss_v = compute_loss_v(data)
        loss_v.backward()
        mpi_avg_grads(ac_v)    # average grads across MPI processes
        vf_optimizer.step()

    # Log changes from update
    kl, ent, cf = pi_info['kl'], pi_info_old['ent'], pi_info['cf']
    logger.store(LossPi=pi_l_old, LossV=v_l_old,
                 KL=kl, Entropy=ent, ClipFrac=cf,
                 DeltaLossPi=(loss_pi.item() - pi_l_old),
                 DeltaLossV=(loss_v.item() - v_l_old))

# Prepare for interaction with environment
start_time = time.time()
o, ep_ret, ep_len = env.reset(), 0, 0

# Main loop: collect experience in env and update/log each epoch
for epoch in range(epochs):
    for t in range(local_steps_per_epoch):
        # a, v, logp = ac.step(torch.as_tensor(o, dtype=torch.float32))
        with torch.no_grad():
            rr = torch.from_numpy(o.copy()).float().to(device)#.unsqueeze(0)
            pi, _ = ac_pi(rr, None)
            a = pi.sample()
            # logp_a = self.pi._log_prob_from_distribution(pi, a)
            logp = pi.log_prob(a)#.sum(axis=-1)
            v = ac_v(torch.as_tensor(o, dtype=torch.float32).to(device))
```

```
            next_o, r, d, _ = env.step(a.cpu().numpy().item())
            ep_ret += r
            ep_len += 1

            # save and log
            buf.store(o, a.cpu().numpy(), r, v.cpu().numpy(), logp.cpu().numpy())
            logger.store(VVals=v.cpu().numpy())

            # Update obs (critical!)
            o = next_o

            timeout = ep_len == max_ep_len
            terminal = d #or timeout
            epoch_ended = t==local_steps_per_epoch-1

            if terminal or epoch_ended:
                if epoch_ended and not(terminal):
                    print('Warning: trajectory cut off by epoch at %d steps.'%ep_len,
flush=True)
                # if trajectory didn't reach terminal state, bootstrap value target
                if epoch_ended:
                    print('epoch_end')
                    # _, v, _ = ac.step(torch.as_tensor(o, dtype=torch.float32))
                    with torch.no_grad():
                        v =ac_v(torch.from_numpy(o).float().to(device)).cpu().numpy()
                else:
                    print('epret :',ep_ret)
                    v = 0
                buf.finish_path(v)
                if terminal:
                    # only save EpRet / EpLen if trajectory finished
                    logger.store(EpRet=ep_ret, EpLen=ep_len)
                o, ep_ret, ep_len = env.reset(), 0, 0


        # Save model
        if (epoch % save_freq == 0) or (epoch == epochs-1):
            logger.save_state({'env': env}, None)
```

```
        # Perform PPO update!
        update()

        # Log info about epoch
        logger.log_tabular('Epoch', epoch)
        logger.log_tabular('EpRet', with_min_and_max=True)
        logger.log_tabular('EpLen', average_only=True)
        logger.log_tabular('VVals', with_min_and_max=True)
        logger.log_tabular('TotalEnvInteracts', (epoch+1)*steps_per_epoch)
        logger.log_tabular('LossPi', average_only=True)
        logger.log_tabular('LossV', average_only=True)
        logger.log_tabular('DeltaLossPi', average_only=True)
        logger.log_tabular('DeltaLossV', average_only=True)
        logger.log_tabular('Entropy', average_only=True)
        logger.log_tabular('KL', average_only=True)
        logger.log_tabular('ClipFrac', average_only=True)
        logger.log_tabular('StopIter', average_only=True)
        logger.log_tabular('Time', time.time()-start_time)
        logger.dump_tabular()
```

```
if __name__ == '__main__':

    hid_sizes = 128
    gamma = 0.999
    seed = 0
    steps = 10000
    epochs = 150
    cpu = 1
    exp_name = "ppo"



    import os
    os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
    mpi_fork(cpu)  # run parallel code with mpi

    from spinup.utils.run_utils import setup_logger_kwargs
    logger_kwargs = setup_logger_kwargs(exp_name, seed)
    # from baselines.common.vec_env.subproc_vec_env import SubprocVecEnv
```

```
    env_fn = lambda : create_train_env(1,1,'complex')
    # env_fn = SubprocVecEnv([])
    # env_fn = lambda : JoypadSpace(gym_super_mario_bros.make("SuperMarioBros-{}-{}-
v0".format(1, 1)), gym_super_mario_bros.actions.COMPLEX_MOVEMENT)
    ppo(env_fn, actor=userActor, critic=userCritic,#core.MLPActorCritic, #gym.make(args.env)
        ac_kwargs=dict(hidden_sizes=hid_sizes), gamma=gamma,
        seed=seed, steps_per_epoch=steps, epochs=epochs,
        logger_kwargs=logger_kwargs, clip_ratio=0.2, pi_lr=0.001, vf_lr=0.001)
```

```
#6.查看训练结果
!pwd
%matplotlib inline
!python -m spinup.run plot /root/lele/spinningup/spinningup/data/ppo/ppo_s0
```

0.0.3 R4.2.1 设计DL模型

```
#导入相关包
import numpy as np
import scipy.signal
from gym.spaces import Box, Discrete

import torch
import torch.nn as nn
from torch.distributions.normal import Normal
from torch.distributions.categorical import Categorical
import torch.nn.functional as F
```

```
#定义通用cnn model作为base model
class cnn_model(nn.Module):
    def __init__(self, num_inputs, num_out, activation=nn.ReLU):
        super(cnn_model, self).__init__()
        self.conv1 = nn.Conv2d(num_inputs, 32, 3, stride=2, padding=1) #卷积层
        self.conv2 = nn.Conv2d(32, 32, 3, stride=2, padding=1)  #卷积层
        self.conv3 = nn.Conv2d(32, 32, 3, stride=2, padding=1)  #卷积层
        self.conv4 = nn.Conv2d(32, 32, 3, stride=2, padding=1)  #卷积层
        self.liner = nn.Linear(32 * 6 * 6, 512) #线性层
        self.fc_out = nn.Linear(512, num_out)   #输出层
```

```
        self._initialize_weights() #模型权重初始化

    #对模型参数进行初始化，合理的初始化对训练的收敛起到非常好的作用
    def _initialize_weights(self):
        for module in self.modules():
            if isinstance(module, nn.Conv2d) or isinstance(module, nn.Linear):
                nn.init.xavier_uniform_(module.weight)
                nn.init.constant_(module.bias, 0)
            elif isinstance(module, nn.LSTMCell):
                nn.init.constant_(module.bias_ih, 0)
                nn.init.constant_(module.bias_hh, 0)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = x.view(x.size(0), -1) #把卷积输出的高纬数据拉平
        x = F.relu(self.liner(x))
        out = self.fc_out(x)
        return out
```

0.0.4 R4.2.2 设计ppo的ac模型（Actor/Critic）

Actor负责policy，输出具体要执行的action， Critic负责预测该状态下的value 指引action更新policy

训练的时候我们通过Critic的指引来更新Actor，而Critic是由Reward指引更新的

```
#定义actor，负责输出action的概率分布，对应简化版的pi函数
class userActor(nn.Module):

    def __init__(self, obs_dim, act_dim, activation):
        super().__init__()
        self.logits_net = cnn_model(obs_dim, act_dim, activation=activation) #定义策略模型，输出为动作空间大小
        print(self.logits_net)
```

```
    #计算策略分布，和action的概率
    def forward(self, obs, act=None):
        pi = Categorical(logits=self.logits_net(obs))
        logp_a = None
        if act is not None:
            logp_a = pi.log_prob(act)
        return pi, logp_a
```

```
#定义Critic，对应简化板的v函数，训练的时候我们通过Critic的指引来更新Actor，而Critic是由Reward指引更新的
class userCritic(nn.Module):

    def __init__(self, obs_dim, activation):
        super().__init__()
        self.v_net = cnn_model(obs_dim, 1, activation=activation) #输出为1，因为输出为当前策略的value预测值（一维）
        print(self.v_net)

    def forward(self, obs):
        return torch.squeeze(self.v_net(obs), -1) # Critical to ensure v has right shape.
```

0.0.5 R4.2.3 定义ppo 算法

```
import numpy as np
import torch
import torch.nn as nn
from torch.optim import Adam
import gym
import time
import scipy.signal
#from core import userCritic, userActor
from env import create_train_env
from spinup.utils.logx import EpochLogger
from spinup.utils.mpi_pytorch import setup_pytorch_for_mpi, sync_params, mpi_avg_grads
from spinup.utils.mpi_tools import mpi_fork, mpi_avg, proc_id, mpi_statistics_scalar, num_procs
```

```
---------------------------------------------------------------------------

ModuleNotFoundError                       Traceback (most recent call last)

<ipython-input-2-efe46b6a5583> in <module>
      1 import numpy as np
----> 2 import torch
      3 import torch.nn as nn
      4 from torch.optim import Adam
      5 import gym

ModuleNotFoundError: No module named 'torch'
```

```
#指定使用gpu
device = torch.device('cuda')
```

```
#utils:
def count_vars(module):
    return sum([np.prod(p.shape) for p in module.parameters()])

def discount_cumsum(x, discount):
    return scipy.signal.lfilter([1], [1, float(-discount)], x[::-1], axis=0)[::-1]
```

```
def combined_shape(length, shape=None):
    if shape is None:
        return (length,)
    return (length, shape) if np.isscalar(shape) else (length, *shape)
```

0.0.5.0.1 R4.2.3.1 定义PPOBuffer 用来存储交互数据，提供给模型训练使用

```
class PPOBuffer:
    """
    A buffer for storing trajectories experienced by a PPO agent interacting
    with the environment, and using Generalized Advantage Estimation (GAE-Lambda)
    for calculating the advantages of state-action pairs.
    """

    def __init__(self, obs_dim, act_dim, size, gamma=0.99, lam=0.95):
        self.obs_buf = np.zeros(combined_shape(size, obs_dim), dtype=np.float32)
        self.act_buf = np.zeros(combined_shape(size, act_dim), dtype=np.float32)
```

```
        self.adv_buf = np.zeros(size, dtype=np.float32)
        self.rew_buf = np.zeros(size, dtype=np.float32)
        self.ret_buf = np.zeros(size, dtype=np.float32)
        self.val_buf = np.zeros(size, dtype=np.float32)
        self.logp_buf = np.zeros(size, dtype=np.float32)
        self.gamma, self.lam = gamma, lam
        self.ptr, self.path_start_idx, self.max_size = 0, 0, size

    def store(self, obs, act, rew, val, logp):
        """
        Append one timestep of agent-environment interaction to the buffer.
        """
        assert self.ptr < self.max_size     # buffer has to have room so you can store
        self.obs_buf[self.ptr] = obs
        self.act_buf[self.ptr] = act
        self.rew_buf[self.ptr] = rew
        self.val_buf[self.ptr] = val
        self.logp_buf[self.ptr] = logp
        self.ptr += 1

    def finish_path(self, last_val=0):
        """
        Call this at the end of a trajectory, or when one gets cut off
        by an epoch ending. This looks back in the buffer to where the
        trajectory started, and uses rewards and value estimates from
        the whole trajectory to compute advantage estimates with GAE-Lambda,
        as well as compute the rewards-to-go for each state, to use as
        the targets for the value function.

        The "last_val" argument should be 0 if the trajectory ended
        because the agent reached a terminal state (died), and otherwise
        should be V(s_T), the value function estimated for the last state.
        This allows us to bootstrap the reward-to-go calculation to account
        for timesteps beyond the arbitrary episode horizon (or epoch cutoff).
        """

        path_slice = slice(self.path_start_idx, self.ptr)
        rews = np.append(self.rew_buf[path_slice], last_val)
        vals = np.append(self.val_buf[path_slice], last_val)
```

```
        # the next two lines implement GAE-Lambda advantage calculation
        deltas = rews[:-1] + self.gamma * vals[1:] - vals[:-1]
        self.adv_buf[path_slice] = discount_cumsum(deltas, self.gamma * self.lam)

        # the next line computes rewards-to-go, to be targets for the value function
        self.ret_buf[path_slice] = discount_cumsum(rews, self.gamma)[:-1]

        self.path_start_idx = self.ptr

    def get(self): #取数据用于训练
        """
        Call this at the end of an epoch to get all of the data from
        the buffer, with advantages appropriately normalized (shifted to have
        mean zero and std one). Also, resets some pointers in the buffer.
        """
        assert self.ptr == self.max_size    # buffer has to be full before you can get
        self.ptr, self.path_start_idx = 0, 0
        # the next two lines implement the advantage normalization trick
        adv_mean, adv_std = mpi_statistics_scalar(self.adv_buf)
        self.adv_buf = (self.adv_buf - adv_mean) / adv_std
        data = dict(obs=self.obs_buf, act=self.act_buf, ret=self.ret_buf,
                    adv=self.adv_buf, logp=self.logp_buf)

        return {k: torch.as_tensor(v, dtype=torch.float32).to(device) for k,v in
data.items()}
```

0.0.5.1 R4.2.3.2 定义ppo算法 及 更新策略

```
#定义policy模型
ac_pi = actor(4, act_dim, activation=nn.Tanh)  # 输入为观察值的channel（4）输出为action的深度（7）
激活函数使用Tanh
```

```
#定义value模型
ac_v = critic(4, activation=nn.Tanh)  # 输入为观察值的channel（4）激活函数使用Tanh
```

```
# Set up function for computing PPO policy loss
def compute_loss_pi(data):
    obs, act, adv, logp_old = data['obs'], data['act'], data['adv'], data['logp']

    # Policy loss
    pi, logp = ac_pi(obs, act) #计算action的概率分布
    ratio = torch.exp(logp - logp_old) #计算新老策略的差异大小
    clip_adv = torch.clamp(ratio, 1-clip_ratio, 1+clip_ratio) * adv #clip 新老策略的更新范围在（1-clip_ratio, 1+clip_ratio）内
    loss_pi = -(torch.min(ratio * adv, clip_adv)).mean() #计算最终的policy loss 使adv（优势）更明显的方向做梯度更新

    return loss_pi
```

```
# Set up function for computing value loss
def compute_loss_v(data):
    obs, ret = data['obs'], data['ret']
    return ((ac_v(obs) - ret)**2).mean() #真实值和预测值做均方差loss 使v模型预测更接近真实值
```

```
# Set up optimizers for policy and value function
pi_optimizer = Adam(ac_pi.parameters(), lr=pi_lr)
vf_optimizer = Adam(ac_v.parameters(), lr=vf_lr)
```

```
#update 模型参数（训练）
def update():
    data = buf.get() #读取训练数据

    # Train policy with multiple steps of gradient descent
    for i in range(train_pi_iters):
        pi_optimizer.zero_grad()
        loss_pi, pi_info = compute_loss_pi(data)
        loss_pi.backward()
        pi_optimizer.step()

    # Value function learning
    for i in range(train_v_iters):
        vf_optimizer.zero_grad()
        loss_v = compute_loss_v(data)
```

```
        loss_v.backward()
        vf_optimizer.step()
```

```
# Prepare for interaction with environment
o, ep_ret, ep_len = env.reset(), 0, 0
# Main loop: collect experience in env and update/log each epoch
for epoch in range(1000):
    for t in range(2000):
        with torch.no_grad(): #收集数据过程，不做参数更新
            rr = torch.from_numpy(o.copy()).float().to(device) #数据转换
            pi, _ = ac_pi(rr, None) # 计算pi
            a = pi.sample() #取出action用于环境交互
            logp = pi.log_prob(a) #去除a对应的概率存起来用于模型训练时判断更新幅度，防止步幅过大学习率过大
            v = ac_v(torch.as_tensor(o, dtype=torch.float32).to(device)) # 计算v存起来给pi模型提供指引

        next_o, r, d, _ = env.step(a.cpu().numpy().item()) #在游戏中执行模型输出的action
        ep_ret += r
        ep_len += 1

        # save and log
        buf.store(o, a.cpu().numpy(), r, v.cpu().numpy(), logp.cpu().numpy())

        # Update obs (critical!)
        o = next_o

        #下面部分为对回合结束时的特殊情况做一下处理，比如最大步数达到了但是并没有gameover则需要获取下一帧的观察值，而小人game over时则不需要，初学者可以掠过
        timeout = ep_len == max_ep_len
        terminal = d
        epoch_ended = t==local_steps_per_epoch-1

        if terminal or epoch_ended:
            if epoch_ended and not(terminal):
                print('Warning: trajectory cut off by epoch at %d steps.'%ep_len, flush=True)
            # if trajectory didn't reach terminal state, bootstrap value target
```

```
            if epoch_ended:
                print('epoch_end')
                with torch.no_grad():
                    v =ac_v(torch.from_numpy(o).float().to(device)).cpu().numpy()
            else:
                print('epret :',ep_ret)
                v = 0
            buf.finish_path(v)
            if terminal:
                # only save EpRet / EpLen if trajectory finished
                logger.store(EpRet=ep_ret, EpLen=ep_len)
            o, ep_ret, ep_len = env.reset(), 0, 0

    # Perform PPO update!
    update()
```

0.0.5.2 R4.2.3.3 PPO算法完整代码（添加log记录、mpi多进程）

```
#ppo函数完整代码
def ppo(env_fn, actor=nn.Module, critic=nn.Module, ac_kwargs=dict(), seed=0,
        steps_per_epoch=4000, epochs=50, gamma=0.99, clip_ratio=0.2, pi_lr=3e-4,
        vf_lr=1e-3, train_pi_iters=80, train_v_iters=80, lam=0.97, max_ep_len=1000,
        target_kl=0.01, logger_kwargs=dict(), save_freq=10):
    # Special function to avoid certain slowdowns from PyTorch + MPI combo.
    setup_pytorch_for_mpi()

    # Set up logger and save configuration
    logger = EpochLogger(**logger_kwargs)
    logger.save_config(locals())

    # Random seed
    seed += 10000 * proc_id()
    torch.manual_seed(seed)
    np.random.seed(seed)

    # Instantiate environment
    env = env_fn()
    obs_dim = env.observation_space.shape
```

```
    act_dim = env.action_space.n

    # Create actor-critic module
    ac_pi = actor(obs_dim[0], act_dim, hidden_sizes=[64, 64], activation=nn.Tanh)  #
env.observation_space, env.action_space, nn.ReLU)
    ac_v = critic(obs_dim[0], hidden_sizes=[64, 64], activation=nn.Tanh)  #
env.observation_space, nn.ReLU)

    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cup')
    ac_pi.to(device)
    ac_v.to(device)

    # Sync params across processes
    sync_params(ac_pi)
    sync_params(ac_v)

    # Count variables
    def count_vars(module):
        return sum([np.prod(p.shape) for p in module.parameters()])
    var_counts = tuple(count_vars(module) for module in [ac_pi, ac_v])
    logger.log('\nNumber of parameters: \t pi: %d, \t v: %d\n'%var_counts)

    # Set up experience buffer
    local_steps_per_epoch = int(steps_per_epoch / num_procs())
    buf = PPOBuffer(obs_dim, env.action_space.shape, local_steps_per_epoch, gamma, lam)

    # Set up function for computing PPO policy loss
    def compute_loss_pi(data):
        obs, act, adv, logp_old = data['obs'], data['act'], data['adv'], data['logp']

        # Policy loss
        pi, logp = ac_pi(obs, act)
        ratio = torch.exp(logp - logp_old)
        clip_adv = torch.clamp(ratio, 1-clip_ratio, 1+clip_ratio) * adv
        loss_pi = -(torch.min(ratio * adv, clip_adv)).mean()

        # Useful extra info
        approx_kl = (logp_old - logp).mean().item()
        ent = pi.entropy().mean().item()
```

```
    clipped = ratio.gt(1+clip_ratio) | ratio.lt(1-clip_ratio)
    clipfrac = torch.as_tensor(clipped, dtype=torch.float32).mean().item()
    pi_info = dict(kl=approx_kl, ent=ent, cf=clipfrac)

    return loss_pi, pi_info

# Set up function for computing value loss
def compute_loss_v(data):
    obs, ret = data['obs'], data['ret']
    return ((ac_v(obs) - ret)**2).mean()

# Set up optimizers for policy and value function
pi_optimizer = Adam(ac_pi.parameters(), lr=pi_lr)
vf_optimizer = Adam(ac_v.parameters(), lr=vf_lr)

# Set up model saving
logger.setup_pytorch_saver(ac_pi)

def update():
    data = buf.get()

    pi_l_old, pi_info_old = compute_loss_pi(data)
    pi_l_old = pi_l_old.item()
    v_l_old = compute_loss_v(data).item()

    # Train policy with multiple steps of gradient descent
    for i in range(train_pi_iters):
        pi_optimizer.zero_grad()
        loss_pi, pi_info = compute_loss_pi(data)
        kl = mpi_avg(pi_info['kl'])
        if kl > 1.5 * target_kl:
            logger.log('Early stopping at step %d due to reaching max kl.'%i)
            break
        loss_pi.backward()
        mpi_avg_grads(ac_pi)    # average grads across MPI processes
        pi_optimizer.step()

    logger.store(StopIter=i)
```

```
    # Value function learning
    for i in range(train_v_iters):
        vf_optimizer.zero_grad()
        loss_v = compute_loss_v(data)
        loss_v.backward()
        mpi_avg_grads(ac_v)    # average grads across MPI processes
        vf_optimizer.step()

    # Log changes from update
    kl, ent, cf = pi_info['kl'], pi_info_old['ent'], pi_info['cf']
    logger.store(LossPi=pi_l_old, LossV=v_l_old,
                 KL=kl, Entropy=ent, ClipFrac=cf,
                 DeltaLossPi=(loss_pi.item() - pi_l_old),
                 DeltaLossV=(loss_v.item() - v_l_old))

# Prepare for interaction with environment
start_time = time.time()
o, ep_ret, ep_len = env.reset(), 0, 0

# Main loop: collect experience in env and update/log each epoch
for epoch in range(epochs):
    for t in range(local_steps_per_epoch):
        # a, v, logp = ac.step(torch.as_tensor(o, dtype=torch.float32))
        with torch.no_grad():
            rr = torch.from_numpy(o.copy()).float().to(device)#.unsqueeze(0)
            pi, _ = ac_pi(rr, None)
            a = pi.sample()
            # logp_a = self.pi._log_prob_from_distribution(pi, a)
            logp = pi.log_prob(a)#.sum(axis=-1)
            v = ac_v(torch.as_tensor(o, dtype=torch.float32).to(device))

        next_o, r, d, _ = env.step(a.cpu().numpy().item())
        ep_ret += r
        ep_len += 1

        # save and log
        buf.store(o, a.cpu().numpy(), r, v.cpu().numpy(), logp.cpu().numpy())
        logger.store(VVals=v.cpu().numpy())
```

```
            # Update obs (critical!)
            o = next_o

            timeout = ep_len == max_ep_len
            terminal = d #or timeout
            epoch_ended = t==local_steps_per_epoch-1

            if terminal or epoch_ended:
                if epoch_ended and not(terminal):
                    print('Warning: trajectory cut off by epoch at %d steps.'%ep_len,
flush=True)
                # if trajectory didn't reach terminal state, bootstrap value target
                if epoch_ended:
                    print('epoch_end')
                    # _, v, _ = ac.step(torch.as_tensor(o, dtype=torch.float32))
                    with torch.no_grad():
                        v =ac_v(torch.from_numpy(o).float().to(device)).cpu().numpy()
                else:
                    print('epret :',ep_ret)
                    v = 0
                buf.finish_path(v)
                if terminal:
                    # only save EpRet / EpLen if trajectory finished
                    logger.store(EpRet=ep_ret, EpLen=ep_len)
                o, ep_ret, ep_len = env.reset(), 0, 0


        # Save model
        if (epoch % save_freq == 0) or (epoch == epochs-1):
            logger.save_state({'env': env}, None)

        # Perform PPO update!
        update()

        # Log info about epoch
        logger.log_tabular('Epoch', epoch)
        logger.log_tabular('EpRet', with_min_and_max=True)
        logger.log_tabular('EpLen', average_only=True)
        logger.log_tabular('VVals', with_min_and_max=True)
```

```
        logger.log_tabular('TotalEnvInteracts', (epoch+1)*steps_per_epoch)
        logger.log_tabular('LossPi', average_only=True)
        logger.log_tabular('LossV', average_only=True)
        logger.log_tabular('DeltaLossPi', average_only=True)
        logger.log_tabular('DeltaLossV', average_only=True)
        logger.log_tabular('Entropy', average_only=True)
        logger.log_tabular('KL', average_only=True)
        logger.log_tabular('ClipFrac', average_only=True)
        logger.log_tabular('StopIter', average_only=True)
        logger.log_tabular('Time', time.time()-start_time)
        logger.dump_tabular()
```

0.0.5.3 R4.2.3.3 主函数

```
if __name__ == '__main__':

    hid_sizes = 128
    gamma = 0.999
    seed = 0
    steps = 10000
    epochs = 150
    cpu = 1
    exp_name = "ppo"


    import os
    os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
    mpi_fork(cpu)  # run parallel code with mpi

    from spinup.utils.run_utils import setup_logger_kwargs
    logger_kwargs = setup_logger_kwargs(exp_name, seed)
    env_fn = lambda : create_train_env(1,1,'complex')
    ppo(env_fn, actor=userActor, critic=userCritic,
        ac_kwargs=dict(hidden_sizes=hid_sizes), gamma=gamma,
        seed=seed, steps_per_epoch=steps, epochs=epochs,
        logger_kwargs=logger_kwargs, clip_ratio=0.2, pi_lr=0.001, vf_lr=0.001)
```

```
--------------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-3-77f82f0b8df1> in <module>
     12     import os
     13     os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
---> 14     mpi_fork(cpu)  # run parallel code with mpi
     15
     16     from spinup.utils.run_utils import setup_logger_kwargs

NameError: name 'mpi_fork' is not defined
```

0.0.5.4 R4.2.3.4 查看训练过程指标

```
#查看训练plot出来
!pwd
%matplotlib inline
!python -m spinup.run plot /root/lele/spinningup/spinningup/data/ppo/ppo_s0
```

0.0.5.5 R4.2.3.5 加载训练好的模型并在游戏中运行

```
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cup')
def load_pytorch_policy(fpath, itr='', deterministic=False):
    """ Load a pytorch policy saved with Spinning Up Logger."""

    fname = osp.join(fpath, 'pyt_save', 'model' + itr + '.pt')
    print('\n\nLoading from %s.\n\n' % fname)

    model = torch.load(fname) #加载训练好的模型

    # make function for producing an action given a single state
    def get_action(x):
        with torch.no_grad():
            x = torch.as_tensor(x, dtype=torch.float32)
            pi, _ = model(x.to(device), None)
            action = pi.sample()
```

```
        return action.cpu()

    return get_action


def run_policy(env, get_action, max_ep_len=None, num_episodes=100, render=True):
    assert env is not None, \
        "Environment not found!\n\n It looks like the environment wasn't saved, " + \
        "and we can't run the agent in it. :( \n\n Check out the readthedocs " + \
        "page on Experiment Outputs for how to handle this situation."

    logger = EpochLogger()

    o, r, d, ep_ret, ep_len, n = env.reset(), 0, False, 0, 0, 0
    while n < num_episodes:
        if render:
            env.render()
            time.sleep(1e-3)

        a = get_action(o)
        o, r, d, _ = env.step(a.numpy().item())
        ep_ret += r
        ep_len += 1

        if d or (ep_len == max_ep_len):
            logger.store(EpRet=ep_ret, EpLen=ep_len)
            print('Episode %d \t EpRet %.3f \t EpLen %d' % (n, ep_ret, ep_len))
            o, r, d, ep_ret, ep_len = env.reset(), 0, False, 0, 0
            n += 1

    logger.log_tabular('EpRet', with_min_and_max=True)
    logger.log_tabular('EpLen', average_only=True)
    logger.dump_tabular()
```

```
if __name__ == '__main__':

    #这里根据你自己的springingup安装路径来修改
    fpath = r'/root/lele/spinningup/spinningup/data/ppo/ppo_s0/'
    episodes = 100
    store_true = False

    env = create_train_env(1,1, 'complx')
    get_action = load_pytorch_policy(fpath)#itr='_50'
    run_policy(env, get_action, 0, episodes, store_true)
```

注：完整代码见：https://github.com/gaoxiaos/Supermariobros-PPO-pytorch.git

0.1 R5 强化学习的近况&挑战

0.0.1 R5.1强化学习的近况

强化学习一直被学术届认为是通往通用智能的大门，所以在这个领域深耕的学术论文每年都在指数增加，特别是今年各大AI会议的论坛都把强化学习的讨论放在了重要位置，比如世界人工智能大会的主论坛、ijcai今年在清华平台举办的麻将AI大赛，nips更是把四个赛道全部放在了强化学习领域

强化学习面临的问题还有很多，比如数据采样的难度，由于数据来源于交互所以模型的学习速度依赖于采样速度，如何在有限的交互步数下取得更好的成绩就成了业内模型创新的方向。再如很多场景无法明确提出奖励函数，这时候如何让模型模仿专家达到专业的程度等各方面的研究都在进行，以及算法方面，自从ppo出来并一统江湖后已经很长时间没有出现质的飞跃的算法，这块的研究也是非常值得进一步探索的。如果你也有兴趣，那么欢迎添加微信入群，和其他小伙伴一起交流探索，打比赛拿奖金。

0.0.2 R5.2强化学习的挑战

通过上面的学习，我们来挑战下有一定难度的新游戏“大鱼吃小鱼”-该游戏是openai新推出的一个“随机生成”环境procgen里非常有代表性的一个场景，为了解决网络模型通常的“记住”怎么走而非完全理解怎么走的问题，openai推出了procgen benchmark用来评估模型的泛化性能，每次reset游戏时游戏的分布都是随机生成的，比如大鱼吃小鱼，每次开场的小鱼分布都是随机的，小鱼行为也都是随机的，这样就需要你控制的angent要真实理解周围环境才能作出正确判断吃到更多小鱼。

竞赛直达：见训练营学习交流群